

AD A 137360

ANALYSIS OF MILITARY
ORGANIZATIONAL EFFECTIVENESS
(AMORE)

PROGRAMMER'S MANUAL

APRIL 1984

Prepared for: U.S. Army Concepts Analysis Agency
8120 Woodmont Avenue
Bethesda, MD 20014

By: SCIENCE APPLICATIONS, INC.
1710 Goodridge Drive
McLean, VA 22102

Contract No: MDA-903-80-C-0409

DTIC FILE COPY

DTIC
ELECTE
JAN 30 1984
B

DISTRIBUTION STATEMENT A

Approved for public release
Distribution Unlimited

84 01 30 02 6



DEPARTMENT OF THE ARMY
US ARMY TRADOC SYSTEMS ANALYSIS ACTIVITY
WHITE SANDS MISSILE RANGE, NEW MEXICO 88002

16 JAN 1984

ATOR-TSL

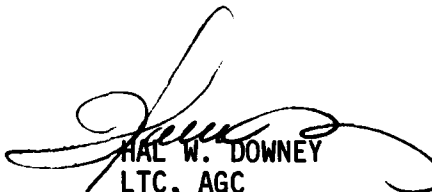
SUBJECT: Letter of Transmittal DTIC System

Defense Technical Information Center
ATTN: DTIC-DDA-2 (Frank Greer)
Cameron Station
Alexandria, Virginia 22314

Request the enclosed document be accepted into the DTIC System with the distribution statement as shown on the accompanying DD Form 1473. The citation is in the DROLS System under ADF050087.

FOR THE DIRECTOR:

1 Encl
as


HAL W. DOWNEY
LTC, AGC
Chief, Support Services Division

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Analysis of Military Organizational Effectiveness (AMORE) Programmer's Manual		5. TYPE OF REPORT & PERIOD COVERED Final Report
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s)		8. CONTRACT OR GRANT NUMBER(s) MDA-903-80-C-0409
9. PERFORMING ORGANIZATION NAME AND ADDRESS Science Applications, Inc. 1710 Goodridge Drive McLean, Virginia 22101		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Commander, USACAA 8120 Woodmont Avenue Bethesda, Maryland 20014		12. REPORT DATE April 1981
		13. NUMBER OF PAGES 157
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Distribution Statement A: approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) AMORE Unit capability		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) The analysis of military organizational effectiveness (AMORE) methodology provides a means for the analysis of unit response to degradation and its recovery of capability over time. The methodology considers both the personnel and equipment of the organization. The interaction of these elements to form teams which contribute to organizational capability is also treated. Following a simulated degradation of the organization, reorganization is accomplished to achieve the maximum capability in the minimum time. The capability, as a function of time following degradation, is provided by exercising the software. Additional		

data is provided for a detailed analysis of the organization's weaknesses as well as its strengths. This manual provides details of the software utilized. A companion User's Manual (ADA111267) provides the analyst/user a basic understanding of the methodology, the unit analysis, and development of input for the software. An updated User's Manual (ADA128045) is also available.

Accession For	
NTIS	✓
DTIC	
DA	
JO	
By	
Dist	
Avail	
Dist	



PREFACE

The purpose of this manual is to provide programming personnel and analysts with details of the AMORE model necessary to effect proper and effective maintenance of the model.

The AMORE model is provided to U.S. Army Concepts Analysis Agency as a part of contract MDA 903-80-C-0409, "Study of Sustainable Loss Rates."

This manual provides a general description of the AMORE model and its structure in Section 1. A more detailed system description is provided in Section 2. Section 3 provides a detailed description with flow charts of each of the subroutines of the model and Section 4 provides a discussion of the operating environment for the UNIVAC system. This manual is intended for use with the corresponding User's Manual, furnished under separate cover.

TABLE OF CONTENTS

<u>SECTION</u>		<u>PAGE</u>
	PREFACE	i
	TABLE OF CONTENTS	ii
	LIST OF FIGURES	iv
	LIST OF TABLES	v
1	GENERAL	1-1
2	SYSTEM DESCRIPTION	2-1
	2.1 General	2-1
	2.2 Dimensioning	2-1
	2.3 Common Blocks	2-7
3	COMPONENT DESCRIPTIONS	3-1
	3.1 INPUTD	3-1
	3.2 Subroutine CUSTMM (NPDSET)	3-9
	3.3 Subroutine INITL (NPDSET)	3-16
	3.4 Subroutine KILL (MP, NN)	3-21
	3.5 Subroutine MAXT (MP, MF, NUMTRY)	3-25
	3.6 Subroutine TRANS (MP, NUMTRY, MF, IS, ILV)	3-27
	3.7 Subroutine CAPT (MP, MF, NUMTRY)	3-52
	3.8 Subroutine ICAP (JMIN, JMAX, ITEAM, NTASK MF, MAX, IS)	3-55
	3.9 Subroutine WHEN (MP, RTN)	3-57
	3.10 Subroutine RCAP (MAX, ITEAM, RTN, NTASK, NT3, TOT, MF, MP)	3-65
	3.11 Subroutine ASN, (MP, MF, NUMTRY)	3-67
	3.12 Subroutine CHOKE (MP, MF, NUMTRY)	3-71
	3.13 Subroutine STAT (TMEAN, TOTCAP, SD, GMEAN, GSD)	3-81
	3.14 Subroutine OUTD (TMEAN, SD, GMEAN, GSD)	3-83
	3.15 Subroutine OUTS	3-89
	3.16 Subroutine PRNT	3-92
	3.17 Subroutine OUTA	3-98
	3.18 Subroutine PRNTS (J, K, L, KOUNT)	3-98
	3.19 Program PARAM	3-105
4	OPERATING ENVIRONMENT	4-1
	4.1 Hardware	4-1
	4.2 Support Software	4-2

TABLE OF CONTENTS (CONT.)

<u>SECTION</u>		<u>PAGE</u>
	APPENDIX A	
A	MUNKRES' ALGORITHM	A-1
	A.1 General	A-1
	A.2 Algorithm Operations	A-3
	A.3 Alternate Optimal Solutions	A-20

LIST OF FIGURES

<u>FIGURE</u>		<u>PAGE</u>
2-1	AMORE Functional Flow Chart	2-2
3-1	Subroutine INPUTD	3-3
3-2	Subroutine COSTMM	3-11
3-3	Subroutine INITL	3-17
3-4	Subroutine KILL	3-23
3-5	Subroutine MAXT	3-26
3-6	Subroutine TRANS	3-29
3-7	Subroutine CAPT	3-54
3-8	Subroutine ICAP	3-56
3-9	Subroutine WHEN	3-58
3-10	Subroutine RCAP	3-66
3-11	Subroutine ASN	3-69
3-12	Subroutine CHOKe	3-74
3-13	Subroutine STAT	3-82
3-14	Subroutine OUTD	3-85
3-15	Subroutine OUTS	3-90
3-16	Subroutine PRNT	3-94
3-17	Subroutine OUTA	3-99
3-18	Subroutine PRNTA	3-101
3-19	Subroutine PARAM	3-107

APPENDIX A

A-1	Tasks (Demands)	A-1
A-2	Munkres' Algorithm	A-4
A-3	The Initial Cost Matrix	A-6
A-4	Cost Matrix Following Step One	A-7
A-5	Cost Matrix Following Step Two	A-8
A-6	Cost Matrix Following Step Three	A-9
A-7	Cost Matrix Following Step Four	A-12
A-8	Cost Matrix Following Step Two A Second Time	A-12
A-9	Cost Matrix Before Entering Step Five	A-13
A-10	Cost Matrix Following Step Five	A-14
A-11	Cost Matrix Following Step Three	A-15
A-12	Cost Matrix Following Step Four	A-15
A-13	Cost Matrix Following Step Two	A-16
A-14	Cost Matrix Following Step Three	A-17
A-15	Cost Matrix Following Step Five	A-17
A-16	Cost Matrix Following Step Three	A-18
A-17	Solution Payoff Matrix Following the Last Iteration of Step Four	A-19
A-18	The Final Cost and Allocation Matrix	A-19
A-19	Original Cost Matrix	A-21
A-20	The Payoff Matrix	A-21

LIST OF FIGURES (CONT.)

<u>FIGURE</u>		<u>PAGE</u>
A-21	Finding a Chain for Reallocation	A-22
A-22	Finding a Chain for Reallocation	A-22
A-23	Finding a Chain for Reallocation	A-23
A-24	Finding a Chain for Reallocation	A-23
A-25	Finding a Chain for Reallocation	A-23
A-26	The Reallocation Chain Completed	A-24
A-27	Alternate Solution	A-24

LIST OF TABLES

<u>TABLE</u>		<u>PAGE</u>
1-1	AMORE Model Hierarchy	1-2

SECTION 1

GENERAL

Program AMORE is written in ANSI FORTRAN and consists of eighteen subprograms and a MAIN routine. The subroutines may be categorized as two input routines, one file handling routine, ten functional routines, and five output routines. The MAIN routine performs no function other than the control of the subroutine operations. The model is constructed in three levels of hierarchy: Level 0, MAIN; Level 1, those subroutines called by MAIN; and Level 2, those subroutines called by a Level 1 subroutine. Table 1-1 provides a display of the model hierarchy along with a brief description of each subroutine function.

The general functioning of the model may be described as follows:

A stochastic assessment of the degradation of assets (personnel and materiel), based on input probabilities of degradation is made.

The surviving assets are then assigned, by use of a transportation algorithm, to satisfy the demands of a particular requirement (team). The largest requirement which can be satisfied is found using a binary search technique. The search results in selection of a particular set of requirements for the application of the transportation algorithm. The cost used by the transportation algorithm is defined as the time required for a particular asset to become operationally effective in some task (the demand). Assignments are made in a manner which will provide a solution with minimum total time cost. Alternate optimal solutions may be examined if desired.

TABLE 1-1 AMORE Model Hierarchy

LEVEL 0 AMORE MAIN		Control of all Processing
<u>LEVEL 1</u>	<u>LEVEL 2</u>	
INPUTD		Read & store input data, except PD's
(Loop each PD set)		
COSTMM		Read PD's and decision times, set up cost arrays. If all PD's read - STOP.
INITL		Initialize storage files for each new PD set
(Loop on Iterations)		
(Loop on Personnel/Materiel)		
KILL		Stochastic damage application - establish survivor arrays
(Loop on Missions)		
MAXT		Establish maximum number of teams that can be constructed (NUMTRY). If dummy resources or invalid transfers are required by TRANS decrease NUMTRY, otherwise increase
	TRANS	Allocate resources to fill the demands of team # - NUMTRY
	CAPT	Determine teams available at times of interest
	ICAP	Determine initial capability teams available from survivors with no transfers.
	WHEN	Determine when transferred and/or repaired assets will be available
	RCAP	Determine the number of teams which can be formed at each time from available assets
(OPTIONAL)		
ASN		Store assignment data for teams built, accumulate data for iterations that result in the same number of teams built

TABLE 1-1 AMORE Model Hierarchy (Cont'd)

<u>LEVEL 1</u>	<u>LEVEL 2</u>	
(OPTIONAL) CHOKE		Store and accumulate choke data from each iteration by team, get data from TRANS allocations for N + 1 teams
	TRANS	Allocate resources, using dummy supply and/or invalid transfers, to construct N + 1 teams
	(OPTIONAL) Entry ALTOPT	Search for other optimal allocations - start from choke point (dummy resource or invalid transfer) only
(End Missions Loop) (End Personnel/Materiel Loop)		
STAT		Accumulate iteration capability statistics for personnel, materiel, and unit by time and mission
(End Iteration Loop)		
OUTD		Calculate average capability and confidence interval for personnel, materiel, and unit; by time and mission; calculate integral of unit capability over time
(OPTIONAL) OUTS		Read stored choke data and control for PRNT
	PRNT	Calculate average needs and surpluses, and their standard deviations for each choke team, Print choke output
(OPTIONAL) OUTA		Read stored assignment data and control for PRNTA
	PRNTA	Calculate average assignments for each team built and print assignment matrix.
(End PD Loop)		

After the maximum requirement which can be met is established, each incremental requirement (team) up to that maximum, is examined to determine when the allocated assets will be available, and, consequently, when each team will be complete. This is done using a random number drawn from an exponential distribution. This process establishes the time that each individual or item is available in its allocated position. This then determines the time when each team is completed.

The next step is to apply the transportation algorithm using the next higher set of requirements (the next team). This serves to identify those assets that are required by, or critical to, the satisfaction of the next higher requirement. Assets which are surplus to that requirement are also identified.

Because of the stochastic processes of the model it is necessary to replicate the above steps several times to derive expected values. Each replication of the damage application may process several sets (missions) of requirements (teams). Processing is also completed for two different types of assets (personnel and materiel). The basic difference in the processing of these assets is that survivors in the materiel category are divided into two additional return categories representing light and moderate (or crew and unit) repair. The fraction of total teams completed at the times of interest is stored for each replication. Additionally, the minimum of the personnel and materiel values at each time, representing the maximum unit capability, is stored. These values are averaged over all iterations and ninety percent confidence intervals are calculated for output after completion of all iterations.

The assignment matrix, allocations made by the transportation algorithm, and choke data, needs and surpluses for the next

higher team, are stored according to the team and mission number. These values are averaged over the number of iterations of occurrence and output when all replications are completed.

SECTION 2

SYSTEM DESCRIPTION

2.1 GENERAL

Figure 2-1 is a flowchart of the AMORE model in terms of the major functions performed. Each of the functions is keyed to the subroutine(s) which perform that function. This figure is also basically a flowchart of the MAIN routine. Details of each subroutine are provided in Section 3.

2.2 DIMENSIONING:

The required dimensioning of all arrays is determined by six of the input variables: NTIMES, the number of times at which capability is to be computed; NTASKS(1) and NTASKS(2), the number of personnel and materiel task entries; the sum of all elements in the array of initial quantities REG(J,I), where I = 1 for personnel, I = 2 for materiel, and J = all task numbers; NMISON, the number of missions; and NTEAMS, the number of teams of the unit. These variables are used to compute a total of 19 parameters required for dimensioning within AMORE. Those parameters are defined below with their uses in dimensioning.

2.2.1 N11MES

The maximum number of times, the input NTIMES must be less than or equal to this value.

Usage: TIMES(N11MES)
TIMEST(N11MES)

2.2.2 N11KS1

The maximum number of personnel task lines, the input NTASK(1) must be less than or equal to this value.

Usage: PERDLY(N11KS1)
PERPD(N11KS1)
TRANP(N11KS1,N11KS1)
COSTP(N11KS1,N11KS1)
ITEAMP(N11AMS,N11SON,N11KS1)

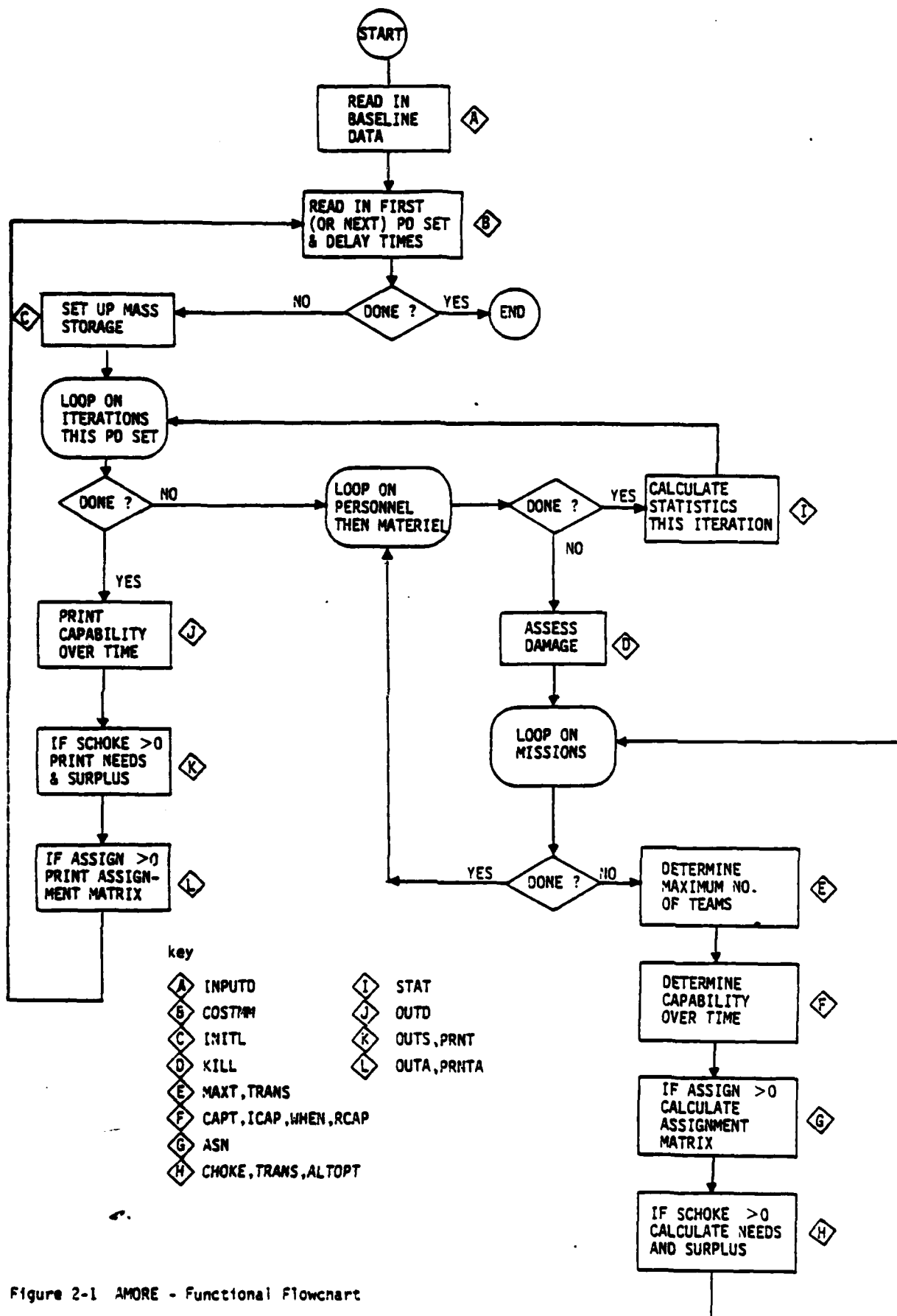


Figure 2-1 AMORE - Functional Flowchart

2.2.3 N11KS2

The maximum number of materiel lines, the input NTASKS(2) must be less than or equal to this value.

Usage: MATDLY(N11KS2)
MATPD(N11KS2,3)
REPTIM(N11KS2,2)
TRANM(N11KS2,N11KS2)
COSTM(N11K9,N11KS2)
ITEAMM(N11AMS,N11SON,N11KS2)

2.2.4 N11NDS

The maximum number total of all personnel and materiel items. (The program PARAM adds 100 to allow some change to authorizations without recompiling the model).

Usage: RAND(N11NDS)

2.2.5 N11SON

The maximum number of missions, the input NMISON must be less than or equal to this value.

Usage: TMEAN(N11K1,N11SON,3)
SD(N11K1,N11SON,3)
TOTCAP(N11K1,N11SON,2)
TOT(N11K1,N11SON,2)
ITEAMP(N11AMS,N11SON,N11KS1)
ITEAMM(N11AMS,N11SON,N11KS2)
ITEAM(N11AMS,N11SON,N11SK1)

2.2.6 N11AMS

The maximum number of teams, the input NTEAMS must be less than or equal to this value.

Usage: ITEAMP(N11AMS,N11SON,N11KS1)
ITEAMM(N11AMS,N11SON,N11KS2)
ITEAM(N11AMS,N11SON,N11SK1)

2.2.7 N11SK1

PERSONNEL

MATERIAL

The larger of N11KS1 or N11KS2.

Usage: AVEN(N11SK1)
ANEED(N11SK1)
MINN(N11SK1)
MAXN(N11SK1)
REG(N11SK1,2)
RETURN(N11K4,N11SK1)
RTN(N11K4,N11SK1)
TASK(N11SK1,3,2)
ITEAM(N11AMS,N11SON,N11SK1)

2.2.8 N11SK3

The larger of N11KS1 or (3 x N11KS2).

Usage: ASURP(N11SK3)
AVES(N11SK3)
IS(N11SK3)
ISOURC(N11SK3)
MAXS(N11SK3)
MINS(N11SK3)

2.2.9 N11K1

N11MES plus 3, the maximum number of times of interest plus zero time, minimum capability time, and infinite time.

Usage: GMEAN(N11K1,N11K2)
GSD(N11K1,N11K2)
TOTCAP(N11K1,N11SON,2)
TOT(N11K1,N11SON,2)
TMEAN(N11K1,N11SON,3)
SD(N11K1,N11SON,3)

2.2.10 N11K2

N11SON minus 1, one less than the maximum number of missions. If (N11SON-1) equal zero, N11K2 is set at 1.

Usage: GMEAN(N11K1,N11K2)
GSD(N11K1,N11K2)

2.2.11 N11K4

N11MES plus 2, all times for capability calculation except zero time.

Usage: RETURN(N11K4,N11SK1)
RTN(N11K4,N11SK1)

2.2.12 N11K9

N11KS2 times 3, count of maximum number of materiel lines plus light and moderate damage categories.

Usage: COSTM(N11K9,N11KS2)

2.2.13 N11K10

N11SK1 plus 1, one more than the larger number of tasks, adds space for dummy demand.

Usage: IALLO(N11K11,N11K10)
MALLO(N11K11,N11K10)
P(N11K11,N11K10)
Z(N11K11,N11K10)
JC(N11K10)
JD(N11K10)
WRK(N11K10)

2.2.14 N11K11

N11SK3 plus 1, one more than the larger of personnel tasks or materiel lines times 3, adds space for dummy supply.

Usage: IALLO(N11K11,N11K10)
ALLO(N11K11,N11K10)
MALLO(N11K11,N11K10)
P(N11K11,N11K10)
Z(N11K11,N11K10)
IR(N11K11)
JR(N11K11)
PS(N11K11,3)

2.2.15 N11K13

(= MRL21)-- N11SK3 times 8, plus 2.

Usage: WORK1(N11K13)

2.2.16 N11K14

N11K10 times N11K11. Used for dimensioning a work storage area of sufficient size for all elements of the allocation arrays.

Usage: WORK(N11K14)

2.2.17 NIDX

N11AMS plus 1, times 2, times N11SON, plus 1. The maximum number of records needed in DEFINE FILE 21 or 22; Choke data for each numbered team and after all teams for both personnel and materiel for all missions.

2.2.18 MRL21

(= N11K13)— N11SK3 times 8, plus 2. The maximum record length for any record of DEFINE FILE 21; for each task plus light and moderate damaged materiel eight data elements may be stored plus the number of iterations (element 1) and total solutions (element 2) considering alternate solutions.

2.2.19 MRL22

The larger of $[N11KS1 \times (N11KS1 + 1)]$ or $[N11SK3 \times (N11KS2 + 1)]$ plus 2. The maximum record length for any record of DEFINE FILE 22; all data elements of the largest possible assignment matrix including the surplus column plus the number of iterations (element 1). Element 2 duplicates element 1.

These 19 dimensioning variables must be available to the AMORE routine in the form of a FORTRAN PROC element GPARAM. GPARAM is INCLUDE'd in every component of the AMORE model.

Program PARAM (para. 3.19) provides the capability to read any data file and calculate the parameters necessary to dimension the AMORE model. PARAM creates the PROC file GPARAM which AMORE accesses by an INCLUDE statement in each routine of the model.

2.3 COMMON BLOCKS

The following common blocks are used in the AMORE model.

2.3.1 DLY1

This common block contains the transfer matrices for both personnel and materiel, the array of repair times for materiel, and the input flag for use of mean or exponentially distributed return times. The variables are: TRANP(N11KS1,N11KS1), TRANM(N11KS2, N11KS2), REPTIM(N11KS2,2), and IMEANT. It is used in the following subroutines: INPUTD, COSTMM, WHEN, and PRNT.

2.3.2 DLY2

The arrays of personnel and materiel delay times are included in this common block. The variables are: (PERDLY(N11KS1) and MATDLY(N11KS2). The arrays are established by subroutine COSTMM and used by subroutine WHEN.

2.3.3 DLY3

This common block contains the cost (total operational delay for transfer) matrices for personnel, COSTP(N11KS1, N11KS1), and materiel, COSTM(N11K9,N11KS2), and the variable NOTEN which is the number of dummy resources required by the transportation algorithm. DLY3 is used by subroutines COSTMM, MAXT, TRANS, CHOK, and WHEN.

2.3.4 GENERL

Common block GENERL contains the number of personnel and materiel tasks, NTASKS(1) and (2); the number of teams, NTEAMS; the array of times of interest, TIMES(N11MES); the number of times, NTIMES; and the number of missions, NMISON. This common block is included in the MAIN routine and all subroutines except ICAP.

2.3.5 INP

The array of task names, TASK(N11SK1,3,2), and the array of initial authorized quantities for each task, REG(N11SK1,2), are contained in this common block. The flag, IONLY, for input processing only is also included. This common block is used by INPUTD, COSTMM, and KILL.

2.3.6 KTR1

Common block KTR1 contains the variables MBIG and MSURP. These variables represent large numbers (calculated by subroutine COSTMM) that are used as costs for non-valid transfers (MBIG) and costs for assignments of surplus or dummy resources (MSURP). The common block KTR1 is included in subroutines COSTMM, MAXT, TRANS, and CHOKE.

2.3.7 KTR2

Common block KTR2 contains the input option flags SCHOKE, ASSIGN, ITRATE, and MULTF. Additionally, the variables ALTAPE, LAST, and IMAX are included. ALTAPE designates the unit number of a scratch file used by INPUTD and TRANS. LAST is the count of alternate solutions found by TRANS(Entry ALTOPT). IMAX is established by a DATA statement in INPUTD and is used by TRANS to insure that no

dimension overflow occurs with some small working arrays. The common block KTR2 occurs in the MAIN routine and in subroutines INPUTD, INITL, TRANS, MAXT, CHOKE, OUTD, OUTS, and PRNT.

2.3.8 PD1

Common block PD1 contains the arrays of damage probabilities for both personnel, PERPD(N11KS1), and materiel, MATPD(N11KS2,3). This common block occurs in subroutines COSTMM and KILL. The arrays are established by COSTMM and used by subroutine KILL.

2.3.9 PD2

Common block PD2 contains the arrays of team requirements for personnel, ITEAMP(N11AMS,N11SON,N11KS1), and materiel, ITEAMM(N11AMS,N11SON,N11KS2). The arrays are established by INPUTD. The common block occurs in INPUTD, TRANS, and CAPT. Additionally, the two arrays are passed as calling arguments from CAPT to both ICAP and RCAP.

2.3.10 PRNTIT

The variables contained in common block PRNTIT are IN, IOUT, IPGCT, LCONT, AND TITLE(20). IN is a variable unit designator for the file from which the input is read. IOUT is the unit designator for the file to which print output is written. IPGCT is the page count and LCONT is the line counter for the print output. TITLE(20) is the 80 character title associated with each input damage set. The common block PRNTIT occurs in the MAIN routine and in subroutines INPUTD, COSTMM, INITL, TRANS, OUTD, OUTS, PRNT, OUTA, and PRNTA.

2.3.11 SEED

Common block SEED contains the seed, ISEED, for the random number generator. ISEED is initialized in MAIN. The common block is necessary for the use of the random number generator BARN and is used in the MAIN routine and subroutines KILL and WHEN.

2.3.12 STATG

Common block STATG contains the arrays TOTCAP(N11K1,N11SON,2) and RETURN(N11K4,N11SK1). TOTCAP contains the calculated capability each iteration for all times and all missions for personnel and materiel. RETURN is an array for each time of interest, except zero time, of the number of individuals or materiel items available for performance in each task. The common block is used in MAIN and CAPT.

2.3.13 STATR

Common block STATR contains the arrays where capability data is accumulated over all iterations. These arrays are: TMEAN(N11K1,N11SON,3), SD(N11K1,N11SON,3), GMEAN(N11K1,N11K2), and GSD(N11K1,N11K2). The common block occurs in MAIN and INITL. The arrays are initialized to zero in INITL prior to beginning any computations for any damage set. The arrays are passed as calling arguments from MAIN to STAT where accumulation occurs over the iterations. They are also passed from MAIN to OUTD, after the completion of all iterations, where average values are computed for TMEAN and GMEAN and confidence intervals are calculated using SD and GSD.

2.3.14 SURV

Common block SURV contains the array of quantities of each task (personnel or materiel) which survive the damage application, array ISOURC(N11SK3). The array is established in subroutine

KILL. The common block occurs in MAIN and subroutines KILL, MAXT, CAPT, and CHOKE. The array is passed from MAXT and CHOKE to TRANS and from CAPT to ICAP as an argument for use by those subroutines.

2.3.15 WK1

Common block WK1 contains the allocation array, IALLO(N11K11, N11K10), which results from the solution of the transportation problem in subroutine TRANS. This common block is included in the MAIN routine and in subroutines TRANS, MAXT, ASN, CHOKE, and WHEN.

2.3.16 WK2

Common block WK2 contains the array WORK(N11K14) which is used by several routines as working storage for a variety of data. The common block occurs in subroutines INITL, TRANS, ASN, CHOKE, OUTS, PRNT, OUTA, and PRNTA.

SECTION 3

COMPONENT DESCRIPTIONS

This section provides flowcharts and general descriptions of the component subroutines of AMORE as follows:

1. Subroutine INPUTD
2. Subroutine COSTMM
3. Subroutine INITL
4. Subroutine KILL
5. Subroutine MAXT
6. Subroutine TRANS
7. Subroutine CAPT
8. Subroutine ICAP
9. Subroutine WHEN
10. Subroutine RCAP
11. Subroutine ASN
12. Subroutine CHOK
13. Subroutine STAT
14. Subroutine OUTD
15. Subroutine OUTS
16. Subroutine PRNT
17. Subroutine OUTA
18. Subroutine PRNTA
19. Program PARAM

3.1 INPUTD

3.1.1 General

Subroutine INPUTD (Figure 3-1) reads the input data from a file designated unit 5, the standard input unit designator in the UNIVAC system. The data undergoes a limited amount of processing and data storage arrays are constructed. Data is formatted for readability and written to Unit 6, the UNIVAC standard print output unit. A scratch file, designated Unit 10, and further referred to as ALTAPE, is also required for use by this subroutine.

Dimensions are checked as data is read to assure array overflows do not occur. The dimensions are established by PARAMETER statements in GPARAM. GPARAM is a PROC file which is constructed by the program PARAM and made available to the program by INCLUDE statements. (See paragraph 2.2.)

Processing is straightforward with the exception of the transfer matrices and team requirements. Input definitions and formats are discussed in detail in the associated User's Manual.

Complexity of processing the transfer matrices is caused by an attempt to simplify the work of user analysts. The first simplification for the analysts was to provide a default of blank to -1. In most cases, this eliminates a large amount of input typing. This was then extended to provide an output of the transfer matrix also void of -1 and with a period (.) substituted for ease of readability. These goals were accomplished by reading the original input using an alpha format. The period is then substituted for both blank and -1 fields and the matrix is printed. The period is then converted to -1 and the matrix is written, as alpha-numeric characters, to file ALTAPE. This file is then read using integer format to obtain a transfer matrix usable by the model.

An effort was also made to reduce the volume of input required to state the team requirements. The technique uses a task multiplier for each requirement input. The multiplier causes the requirement to be repeated for that number of task lines. Each requirement for each task is input as the additional number required for that team. These requirements are converted to cumulative requirements by team for use within the model.

INPUTD

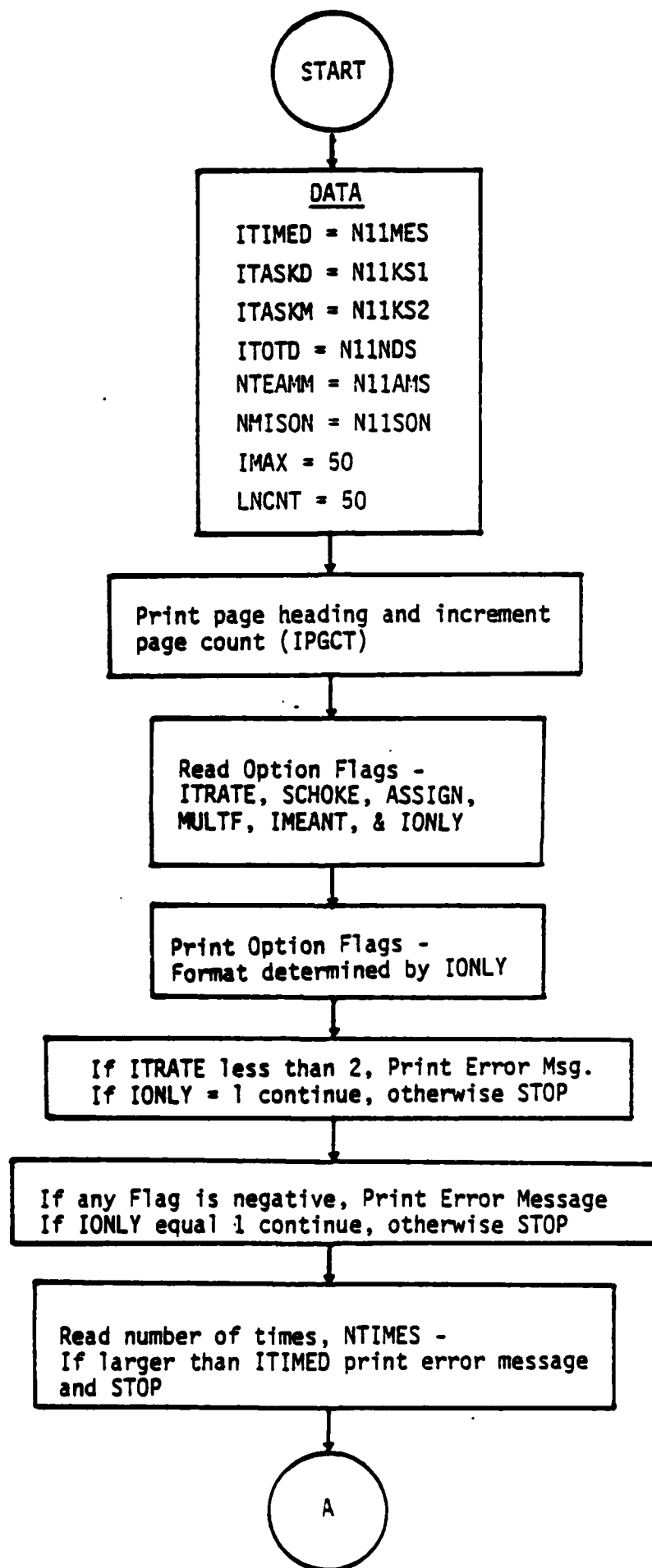


Figure 3-1. Subroutine INPUTD

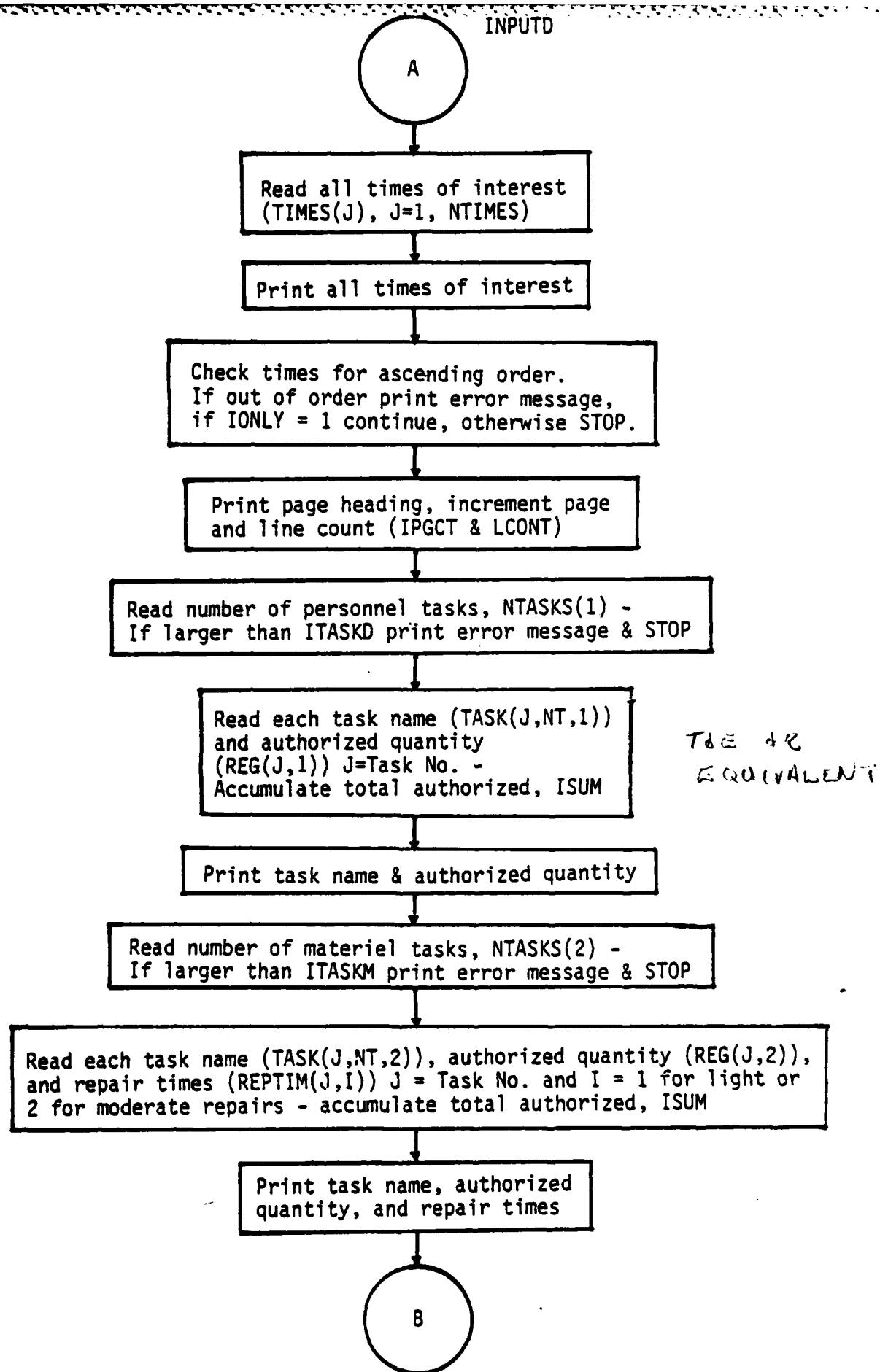


Figure 3-1. Subroutine INPUTD (Continued)

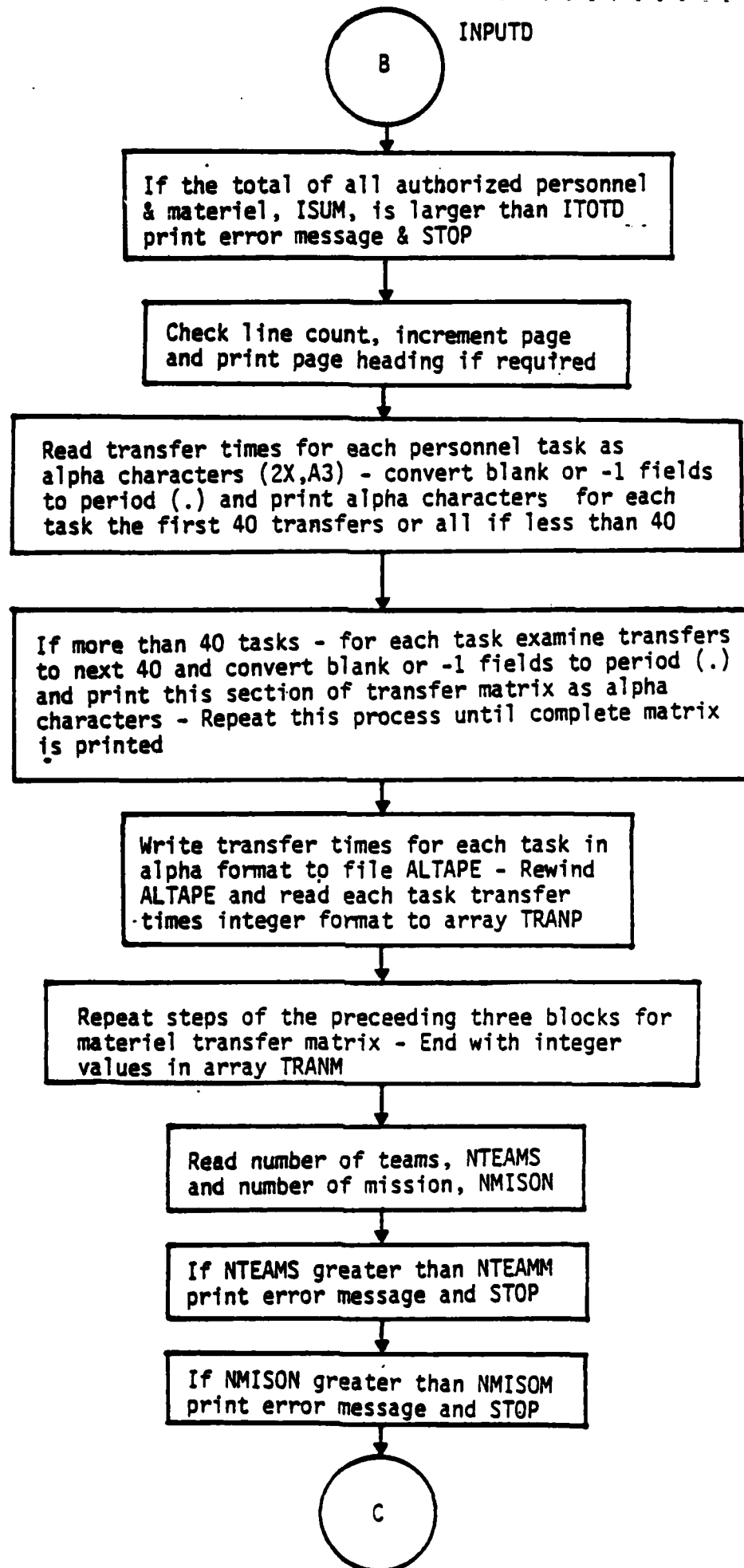


Figure 3-1. Subroutine INPUTD (Continued)

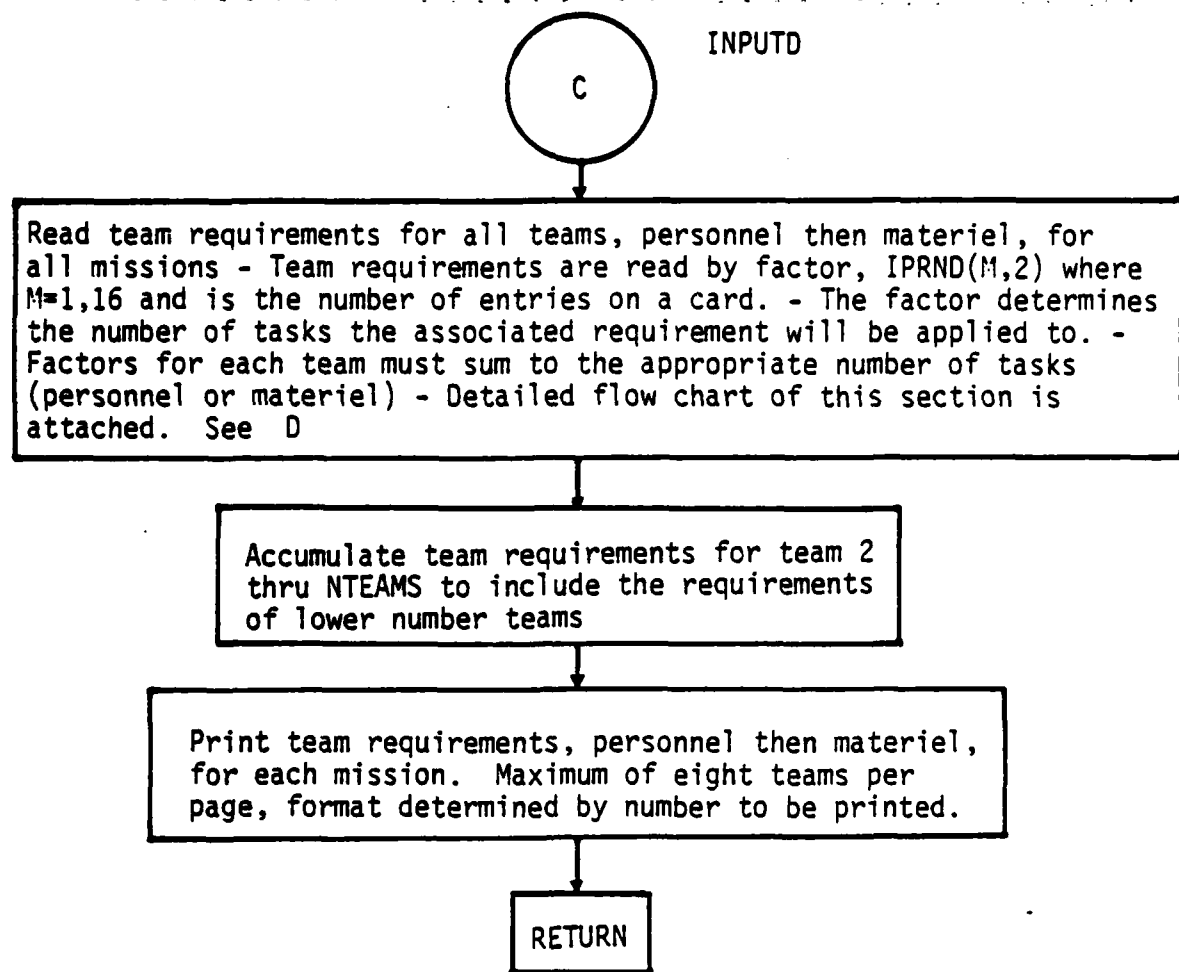


Figure 3-1. Subroutine INPUTD (Continued)

SECRET



Figure 3-1. Subroutine INPUTD (Continued)

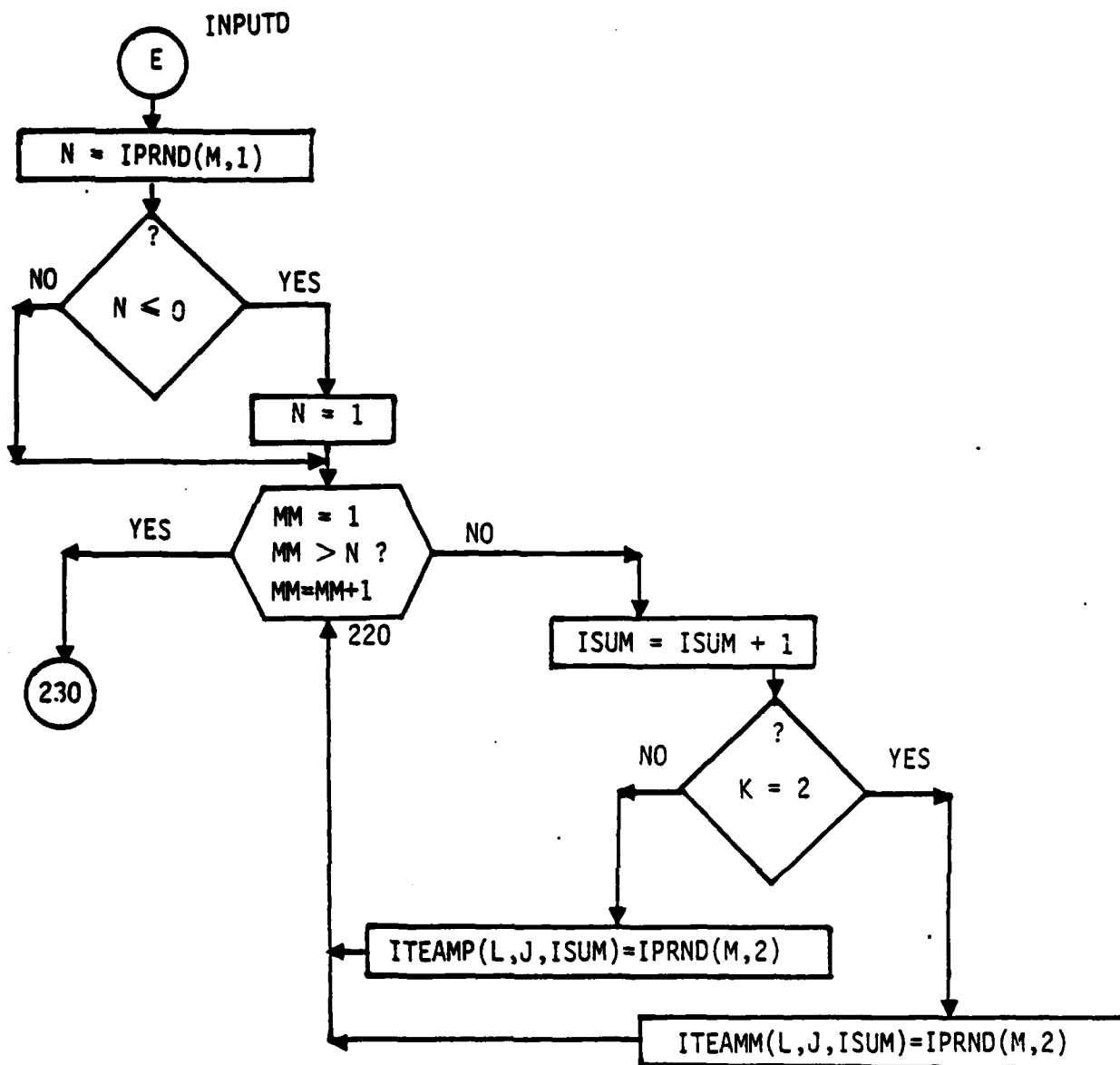


Figure 3-1. Subroutine INPUTD (Continued)

3.1.2 COMMON BLOCKS

DLY1
GENERL
INP
KTR2
PD2
PRNTIT

3.2 SUBROUTINE COSTMM (NPDSSET)

3.2.1 General

Subroutine COSTMM (Figure 3-2) is referenced for each set of degradation probability values (PDSET) as a cost initialization routine. For each individual PDSET, cost matrices are calculated for both personnel and materiel categories.

The TITLE array, an array of alphanumeric title words, is read from the input file for the first PDSET. (If an 'END OF FILE' is encountered while attempting to process this read, control is transferred to a program STOP. Therefore this input statement acts as the normal termination check for the AMORE program model). The TITLE array is then echoed to the output page and determination of the personnel probability of damage array begins.

Initially all array elements of PERPD are set to -1.0. The first input record of damage information is then read. The probability of damage (TEMPPD(1)) is read, followed by the commander's decision time to assess damage and initiate recovery (ITEMPD). On the same input line an index array (INDEX(j)) of as many as 14 values/line may follow ITEMPD. The elements of this index array correspond to personnel task numbers. In this manner, indices of particular tasks are identified with the ITEMPD and TEMPPD(1) value of the same input line. (An alternative: When each of the personnel task shares both

TEMPPD(1) and ITEMPD with all other personnel items, as single '-1' can be substituted in place of the entire input INDEX array.) Subsequent lines are read until all personnel items have been assigned a probability of damage value. These values are stored in array PERPD. Corresponding delay times are stored in array PERDLY. Any attempt to reassign indices or to assign indices of personnel tasks that do not exist will result in an error STOP.

The above arrays are then formatted and printed for the particular PDSET.

Similarly, materiel item probabilities of degradation and delay time (TEMPPD(1), (2) and (3) and ITEMPD respectively) are read from the input file. The element TEMPPD(1) is the total probability of damage, all categories. TEMPPD(2) is the probability of moderate and severe damage, TEMPPD(3) is the probability of severe damage. These values are stored in array MATPD according to indices read from the same card. The value ITEMPD is likewise stored in array MATDLY. The results are then printed.

The next step in COSTMM processing is to check if the current run is of an input-only type (IONLY.GE.1). If this is the case, additional initialization need not be performed and control loops back to read the next PDSET values. Otherwise, certain maximal values must be calculated. The largest repair time within mission horizon time (LPAIR), the largest delay time (LDELAY), the largest transfer time (LTRAN) are determined from existing values. The variable MSURP is calculated by summing these values, multiplying by two and adding one. This value is used in TRANS as the cost for assignment of dummy supply or for assignment as surplus. This large value makes any feasible transfer preferable. Another large cost (MBIG) is then calculated using MSURP. MBIG serves as a large dummy cost when a transfer is infeasible. It is a value larger than the total cost could be if all transfers made had the largest possible cost.

COSTMM

(READ AND WRITE MATERIEL DAMAGE PROBABILITIES)

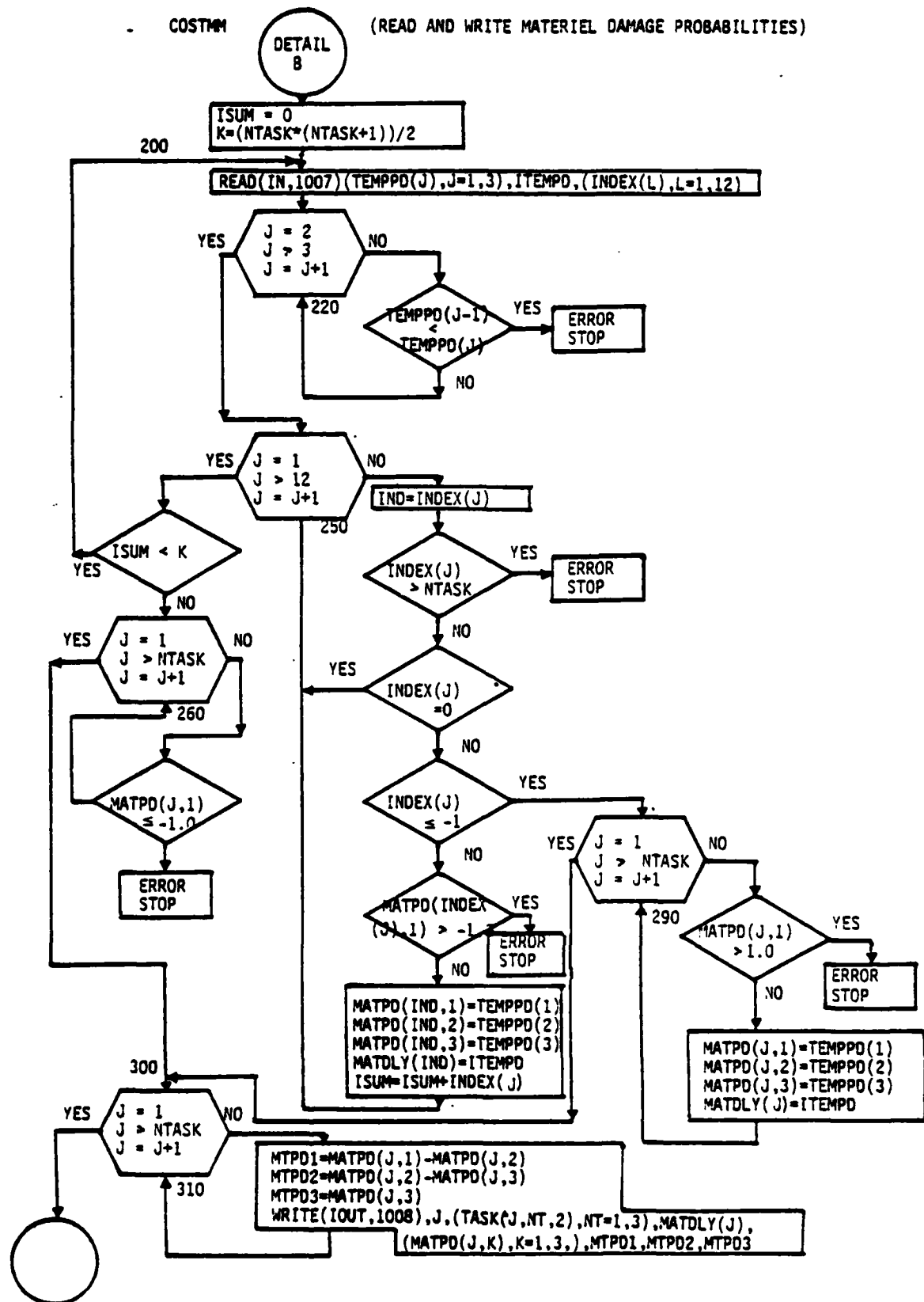


Figure 3-2. Subroutine COSTMM (Continued)

COSTMM

(CONSTRUCT PERSONNEL COST MATRIX)

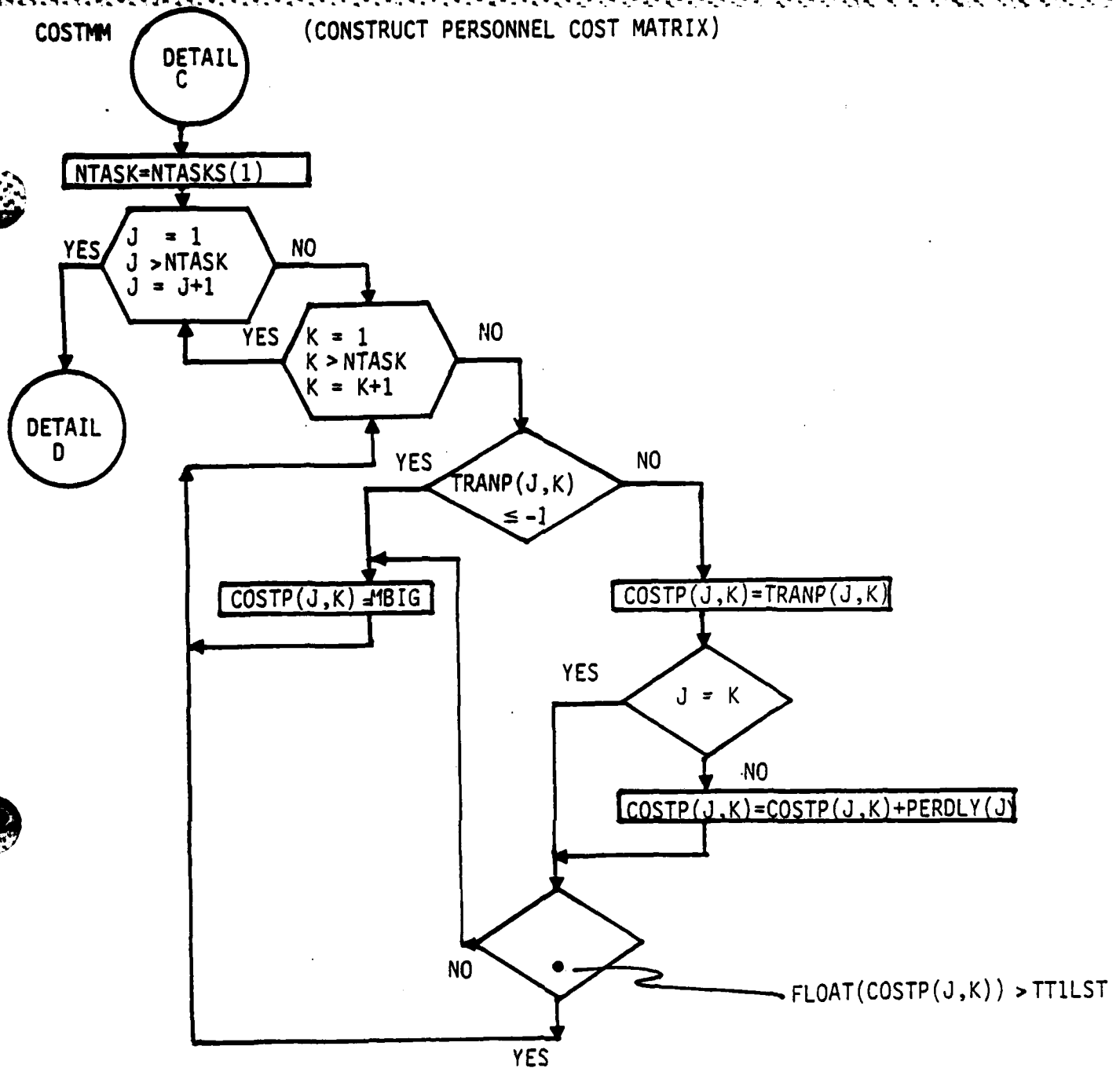


Figure 3-2. Subroutine COSTMM (Continued)

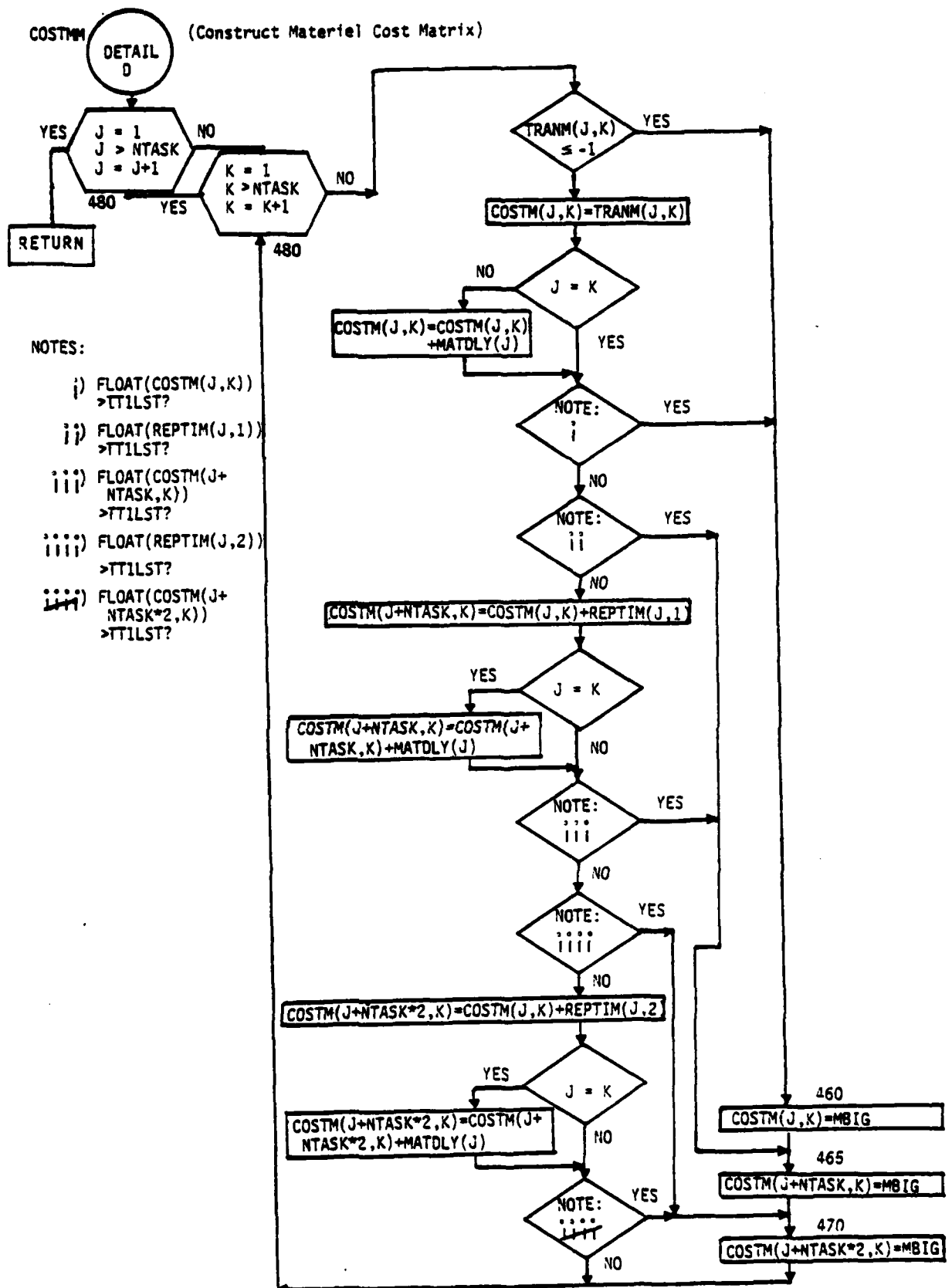


Figure 3-2. Subroutine COSTM: (Continued)

The cost matrix for personnel items (COSTP) is developed from the input transfer matrix (TRANP) and delay time array (PERDLY). (Any unfeasible transfers are assigned a cost of value MBIG.)

Similarly, a materiel cost matrix is built (COSTM). However when calculating materiel costs, repair times for light damage (REPTIM (j, 1)) and those for moderate damage (REPTIM(j, 2)) must be included and the array, COSTM, is extended with separate sections for light and moderate damaged equipment.

3.2.2 COMMON BLOCKS

DLY1
DLY2
DLY3
GENERL
INP
KTR1
PD1
PRNTIT

3.3 SUBROUTINE INITL (NPDSET)

3.3.1 General

Subroutine INITL (Figure 3-3) is used to initialize certain arrays and work files to zero values prior to the start of processing.

Initially, the statistical arrays for Subroutine STAT are set to zero (TMEAN, SD, GMEAN, GSD). The work array elements, following determination of an adequate maximal number of the elements (I), are also set to zero.

If the choke flag indicates that choke data is not desired for this particular run (SCHOKE \leq 0), then choke variable initialization may be omitted. Otherwise both materiel and personnel record length

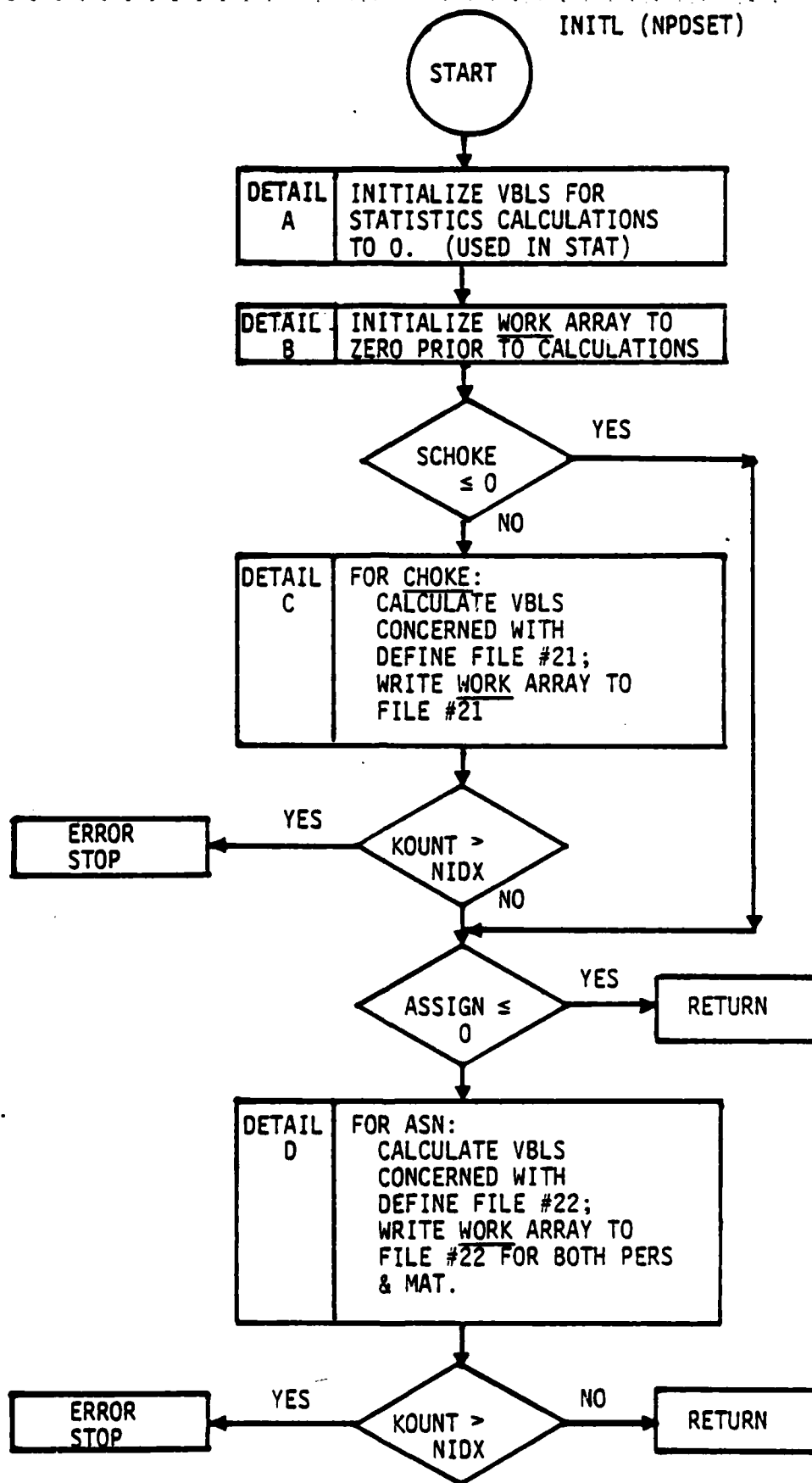


Figure 3-3. Subroutine INITL

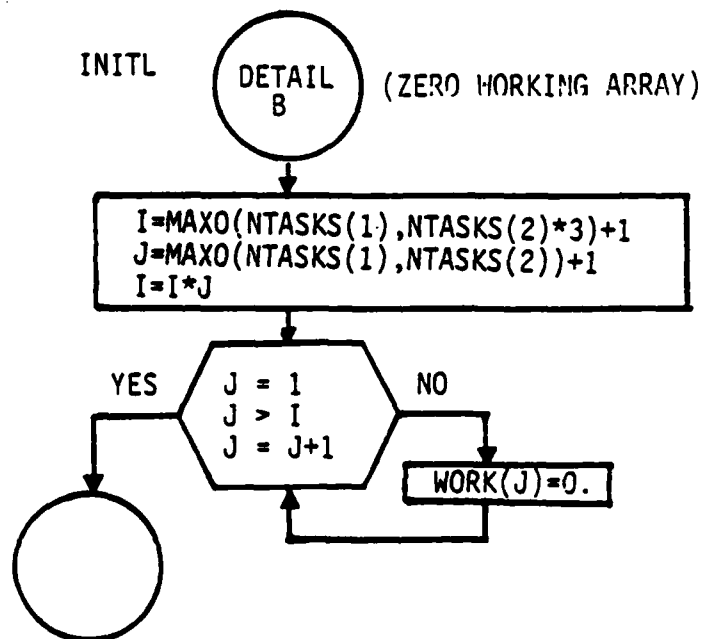
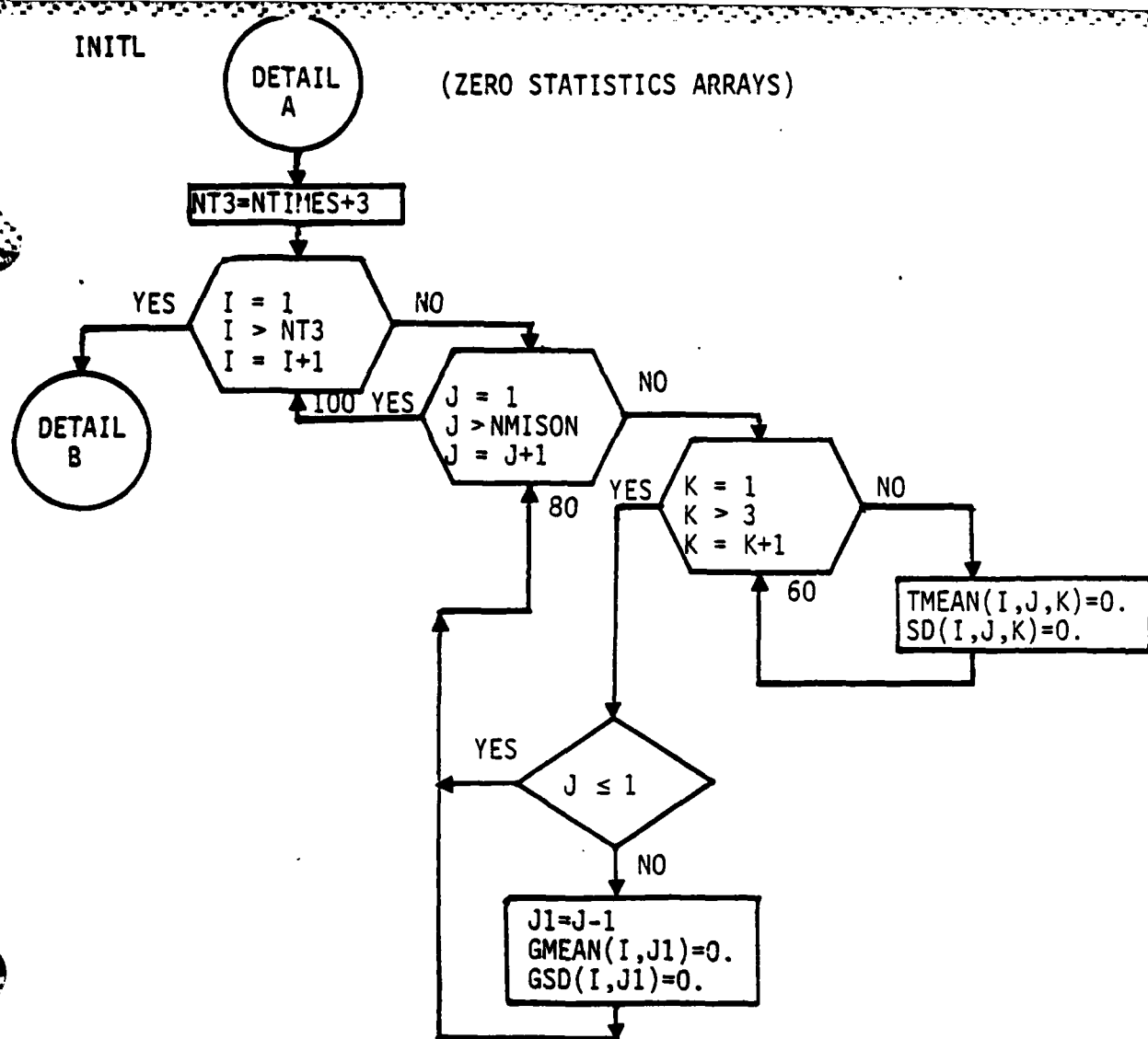


Figure 3-3. Subroutine INITL (Continued)

INITL

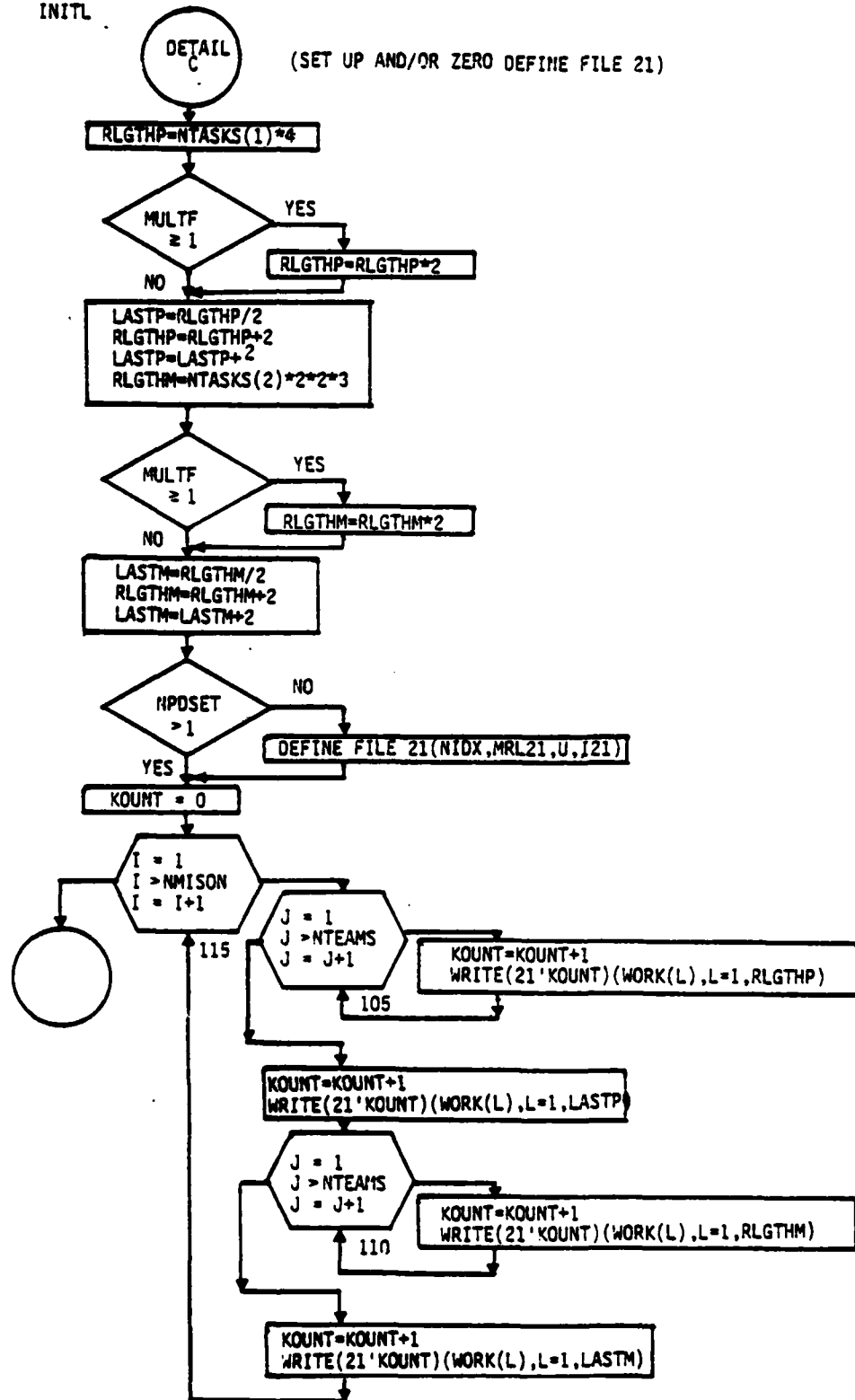


Figure 3-3. Subroutine INITL (Continued)

INITL

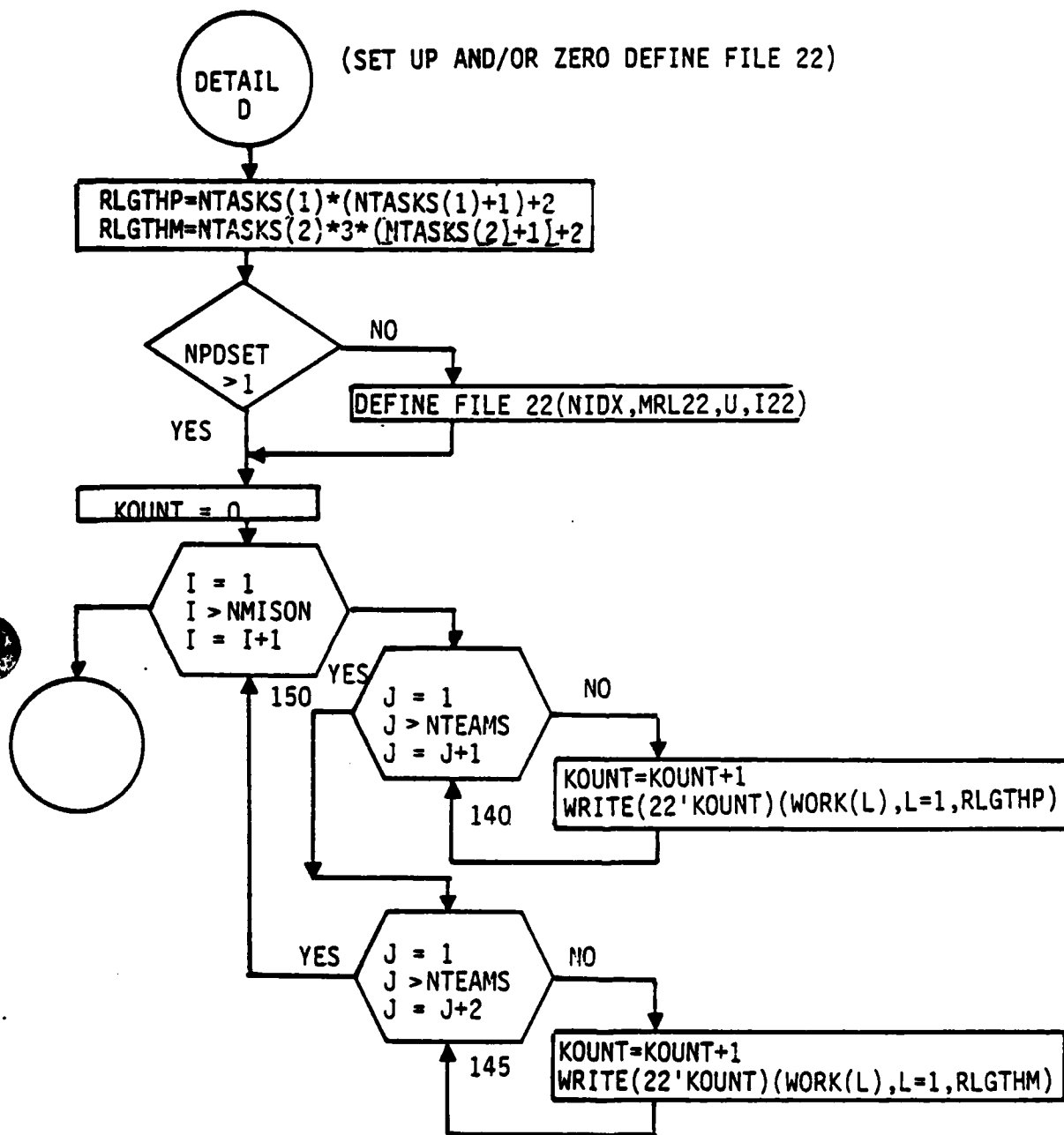


Figure 3-3. Subroutine INITL (Continued)

constraints (RLGTHP, LASTP, RLGTHM, LASTM) are determined for the choke file. This file (#21) is then defined and all elements of all records are zeroed.

If the assignment flag indicates that no assignment data is to be output for this run ($ASSIGN \leq 0$), then assignment data initialization may be omitted. Otherwise record length constraints (RLGHTP, RLGTHM) are determined for the assignment file. The file (#22) is defined and all elements of all records are zeroed.

3.3.2 COMMON BLOCKS

GENERL
KTR2
PRNTIT
STATR
WK2

3.4 SUBROUTINE KILL (MP, NN)

3.4.1 General

Subroutine KILL (Figure 3-4) provides the stochastic application of the input damage probabilities. The subroutine is called by the MAIN routine on each iteration, once for personnel and again for materiel. The argument MP defines either personnel (1) or materiel (2). The argument NN is the iteration number and is used for the application of a variance reduction technique. The variance reduction technique is to draw new random numbers on each odd numbered iteration and to use the complement of those random numbers on the following even numbered iteration.

The subroutine consists of two sections, personnel and materiel, which perform the same basic functions. The initial authorized quantity for each task or materiel line, in array REG, is put into the

survivors array, ISOURC. For each individual or item in this quantity a random number (RANDOM) is obtained. (On odd numbered iterations the random number is obtained from the function BARN and the complement is stored in array RAND. On even numbered iterations the random number is drawn from the array RAND.) RANDOM is then compared to the probability of damage for this item. If the random number is larger than the damage probability, this is a survivor. If RANDOM is smaller, then one is subtracted from the survivors array. In the materiel case, this comparison is made against the accumulated probabilities of light, moderate, and severe damage (MATPD(I,1), where I is the line number). If the item is assessed as damaged then further comparisons are made of the random number with the accumulated probabilities of moderate and severe damage (MATPD(I,2)). If RANDOM is larger than this value, the item is assessed as light damage and is added into the light damage section of the survivors array. If RANDOM is equal or smaller than this last value, it is compared to the probability of severe damage (MATPD(I,3)). If RANDOM is equal to or smaller than MATPD(I,3), the damage is assessed as severe, otherwise it is assessed as moderate damage and is added into the moderate damage section of the survivors array.

Damage is assessed against each individual or materiel item in each task or line. The array of personnel or materiel survivors, ISOURC, in common block SURV results.

3.4.2 COMMON BLOCKS

GENERAL
INP
PD1
SEED
SURV

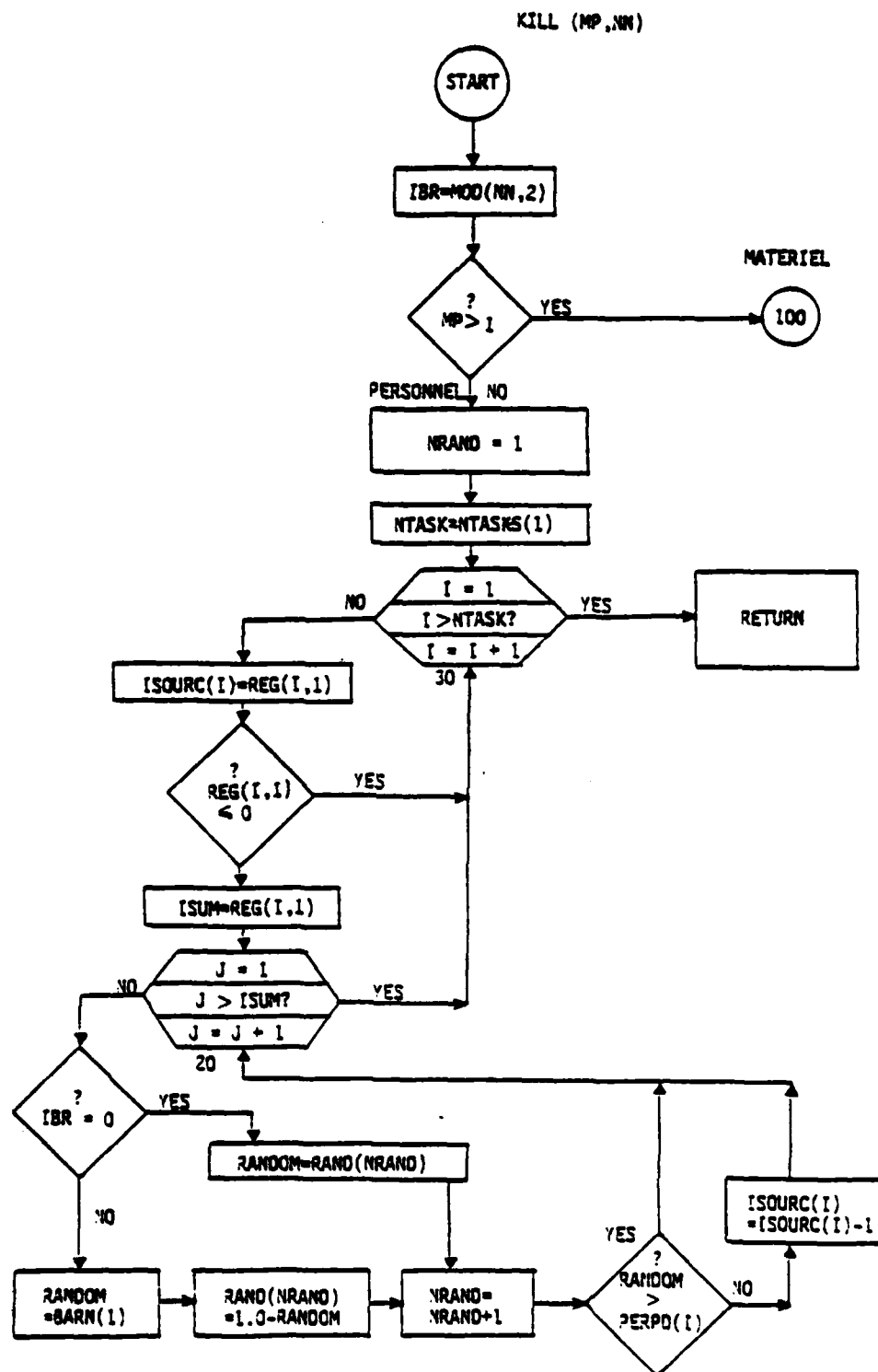


Figure 3-4. Subroutine KILL

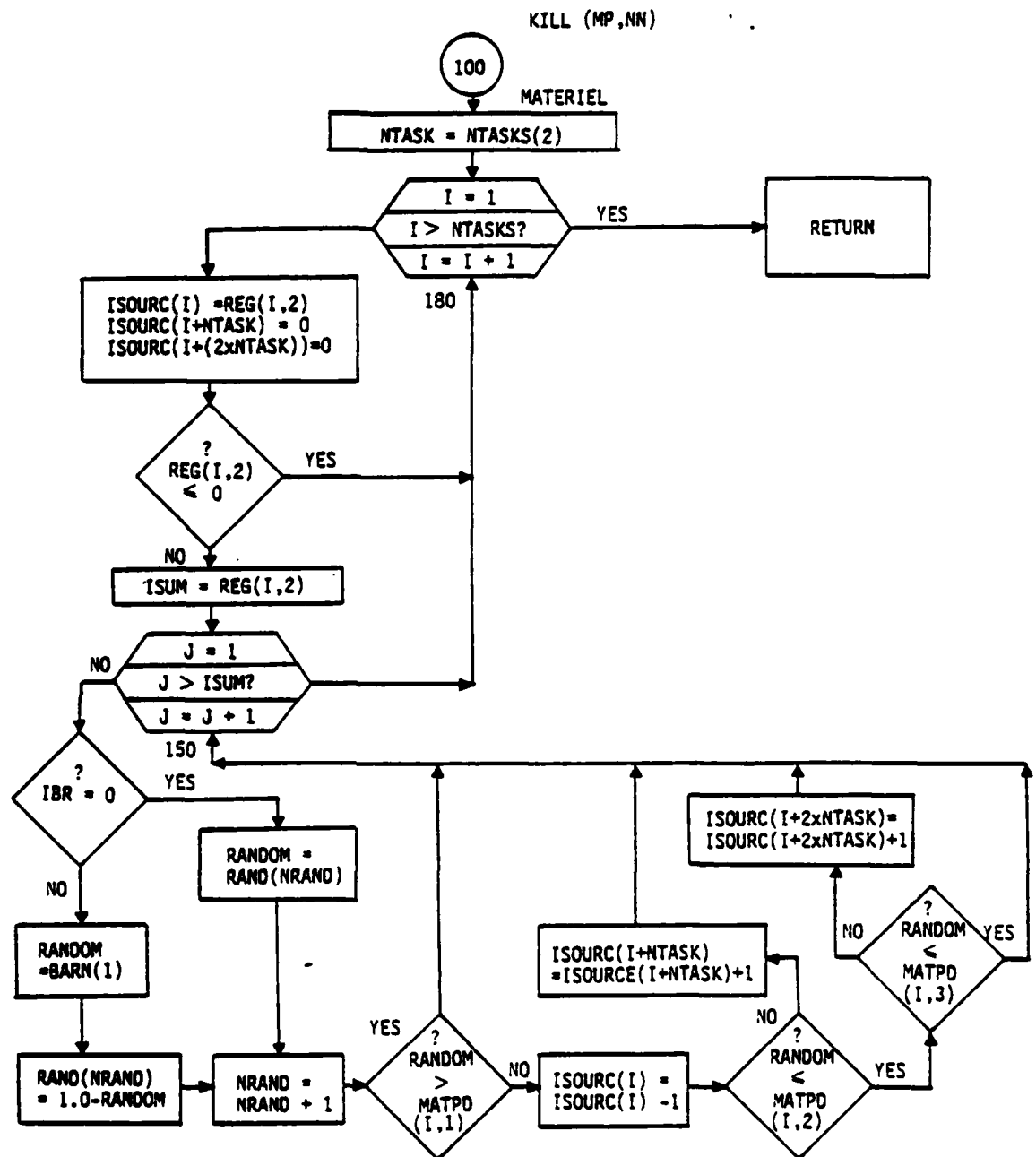


Figure 3.4. Subroutine KILL (Continued)

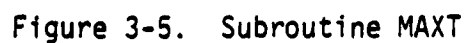
3.5 SUBROUTINE MAXT (MP, MF, NUMTRY)

3.5.1 General

Subroutine MAXT (Figure 3-5) determines the maximum number of teams which can be constructed using the surviving resources. This is accomplished using a binary search technique to vary the team number. Subroutine TRANS is called to determine if a feasible solution is possible with the existing resources. Subroutine MAXT is included within the innermost loop (missions) of the MAIN routine. The arguments MP and MF distinguish between personnel and materiel and identify the mission number, respectively. The argument NUMTRY provides MAIN with the number of teams which could be constructed.

Subroutine MAXT begins its process with a team number at the mid-point of the range, 1 to NTEAMS. The call is made to subroutine TRANS which allocates the surviving resources to the requirements of this team. TRANS will always provide a solution by either adding dummy resources or by making infeasible allocations. On input these infeasible transfers are identified by a minus one (-1) in the transfer matrix. For use in the transportation algorithm they are given a very large cost (MBIG) and are therefore the last recourse for a solution to the allocation problem. MAXT determines the feasibility of the solution by examining the requirement for dummy supply and the cost of the allocations made. NOTEN is the variable which is equal to the dummy supply required. If NOTEN is greater than zero the solution is not feasible. If this check shows a possible feasible solution, then the cost of each allocation is examined. If the cost of any allocation is greater than or equal to MBIG the solution is not feasible.

MAXT then adjusts the team number (NUMTRY) for the next try. If the solution was feasible a higher number team is tried. If the



solution was not feasible a lower team is attempted. After each iteration of this process JMIN is the number of the last feasible team and JMAX is the number of the last infeasible team. The next attempt is made for the team number midway between these two. During this process the variable ILV, last in the argument list for the call to subroutine TRANS, is set to one (1). This variable flags TRANS to return immediately if the solution is proved to be infeasible, such as a requirement for dummy supply. This reduces the number of times that the full transportation algorithm is solved. The transportation algorithm is always solved if there is a possibility of a feasible solution. When the difference between JMIN and JMAX is one, JMIN is the number of the highest team with a feasible solution. If the last solution was feasible, control is returned to MAIN. If the last solution was not feasible then TRANS is called once again, with ILV=0, to provide the solution for that team, number = JMIN.

3.5.2 COMMON BLOCKS

DLY3
 GENERL
 KTR1
 KTR2
 SURV
 WK1

3.6 SUBROUTINE TRANS (MP, NUMTRY, MF, IS, ILV)

3.6.1 General

Subroutine TRANS (Figure 3-6) is an application of the Munkres algorithm to solve an allocation problem. (Munkres algorithm is discussed in depth in Appendix A.) This subroutine is called by subroutine MAXT and by Subroutine CHOKE. It contains one Entry Point, ALTOPT, which is entered only by a call from Subroutine CHOKE. Subroutine TRANS allocates the surviving assets to satisfy the demands

of a particular team and mission requirement. These requirements are established by the calling subroutine through the arguments: MP, personnel (1) or materiel (2); NUMTRY, the team number (1 to maximum); and MF, mission number (1 to n). The arguments IS is the array of surviving assets. The argument ILV is a flag which is used to reduce the number of times the transportation problem is completely solved. If ILV is zero the transportation algorithm will be completed. If ILV equal 1, quick checks are used to determine if a feasible solution is possible. If the total demand exceeds the total resources or if the demand for any one task line exceeds all possible resources for that line, no solution is feasible. During the search to find the maximum possible teams it is not necessary to complete the transportation solution if either of the conditions exists. Other calls set ILV=0 and a full solution is found.

Subroutine TRANS will always provide a solution to the allocation problem. It does this by creating a dummy demand if assets exceed requirements or by creating a dummy supply if requirements are greater than the assets. The routine may also provide a solution by making assignments which are considered to be infeasible. These assignments are made using a very large cost (MBIG) which readily identifies them. MAXT determines if the solution is, in fact, feasible or not based on the use of dummy supply and/or a cost greater than or equal to MBIG. The call from CHOKE results in an infeasible solution in all cases except when all teams can be built.

Entry point ALTOPT is entered only by a call from CHOKE when the option flag MULTF is greater than zero. This part of subroutine TRANS searches the original allocations for alternate optimal solutions. Alternate assignments are examined only for the 'CHOKE' points, that is allocations of dummy supply (Cost = MSURP) or allocations made with cost MBIG.

TRANS (MP, NUMTRY, MF, IS, ILV)

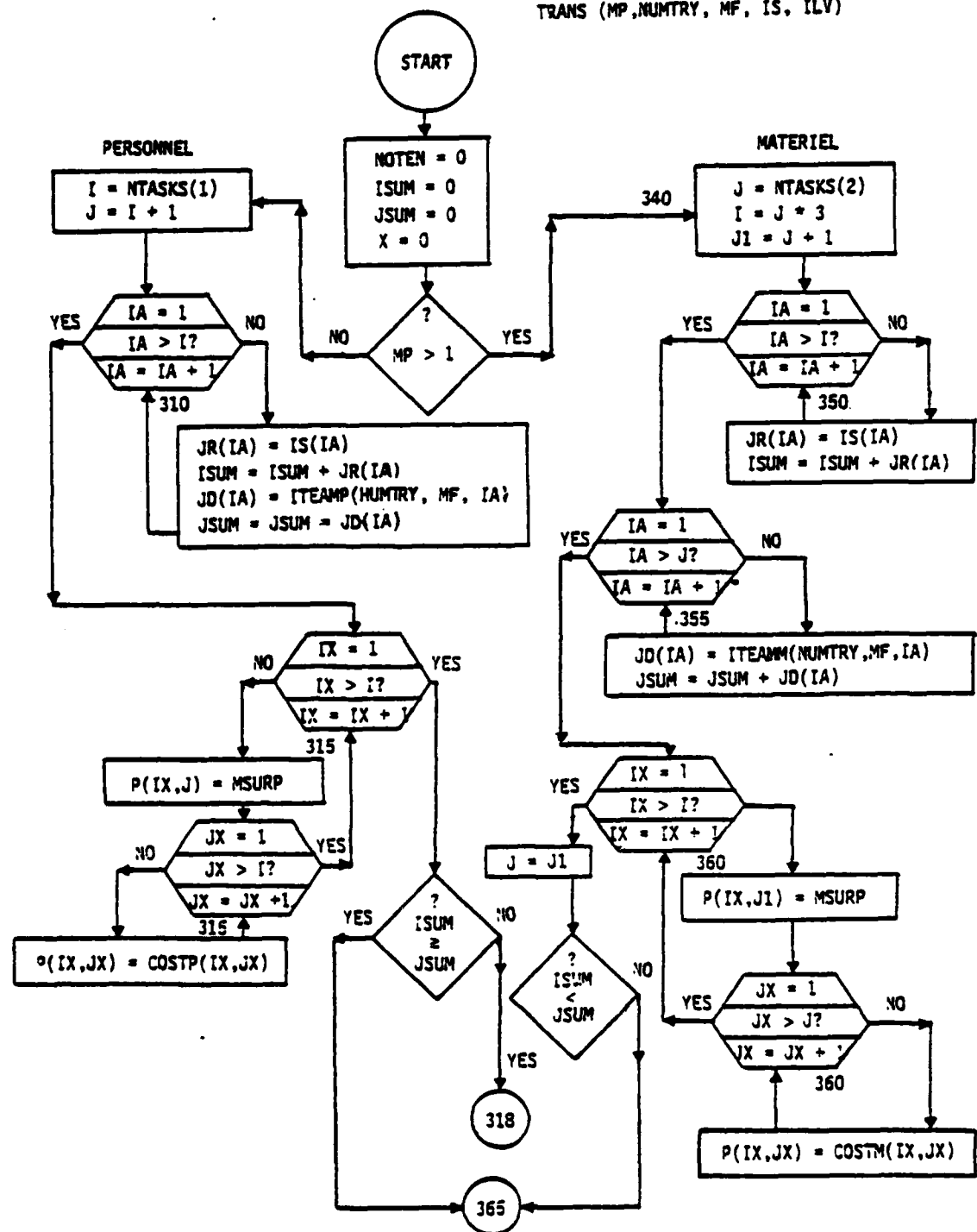


Figure 3-6. Subroutine TRANS

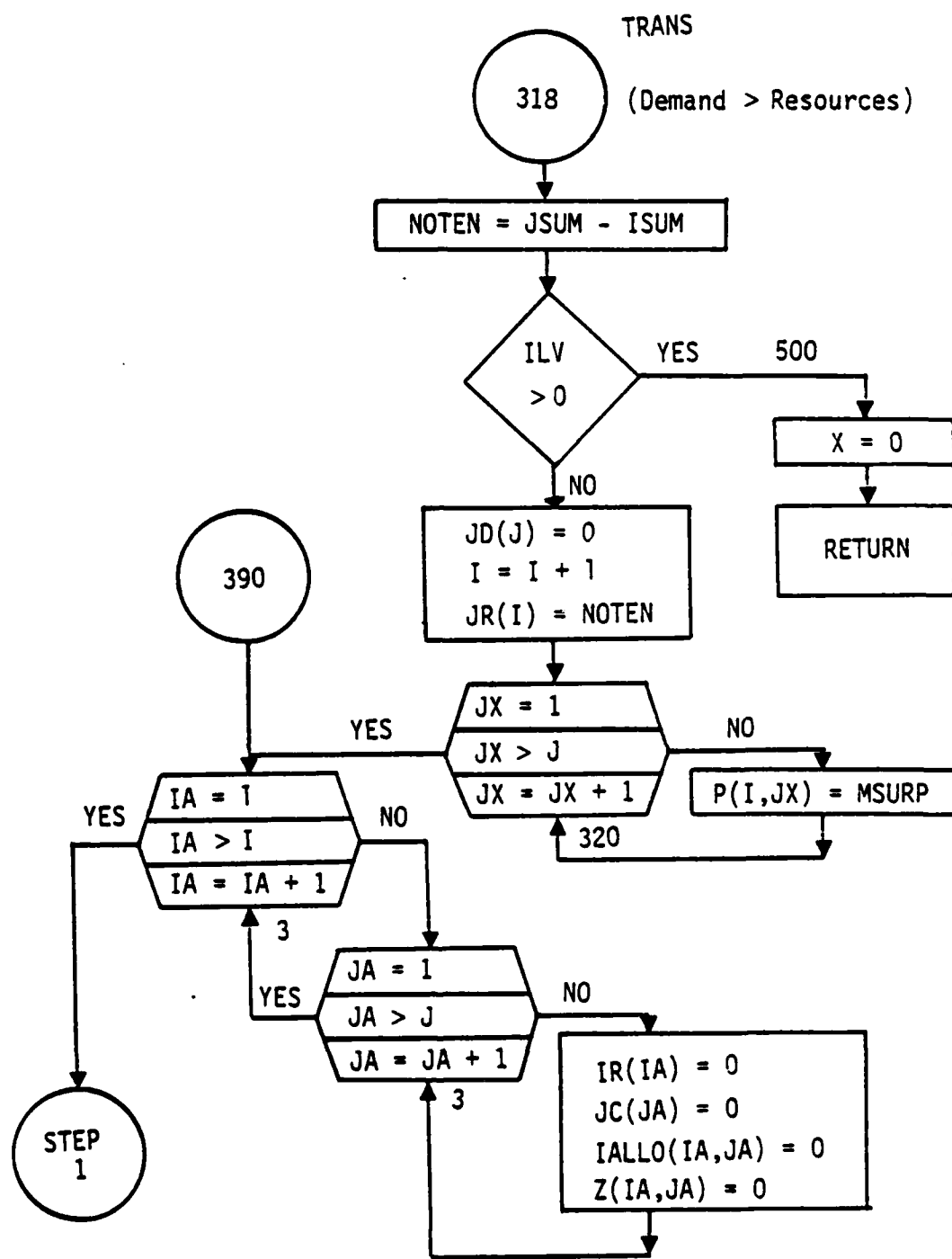


Figure 3-6. Subroutine TRANS (Continued)

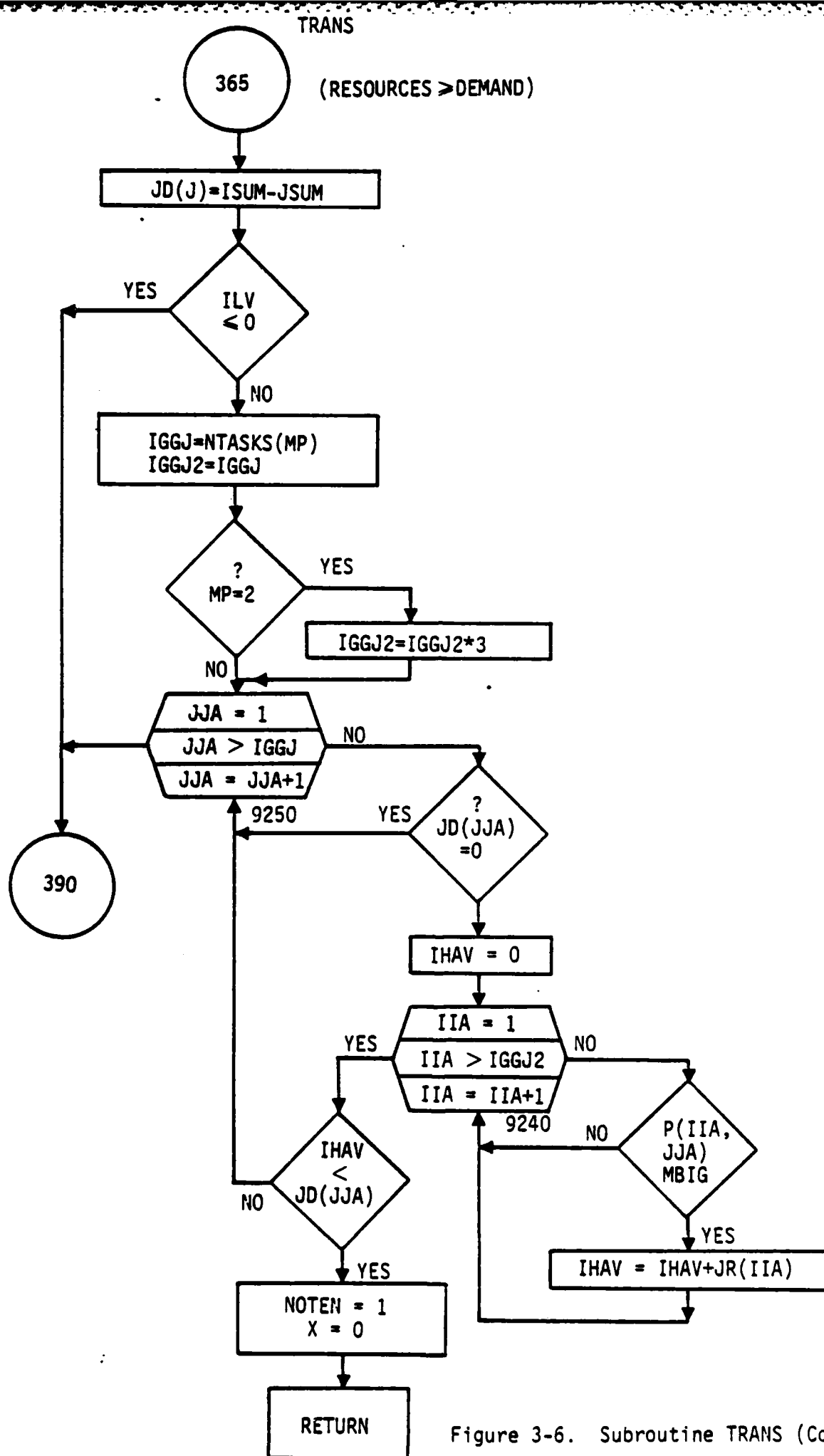


Figure 3-6. Subroutine TRANS (Continued)

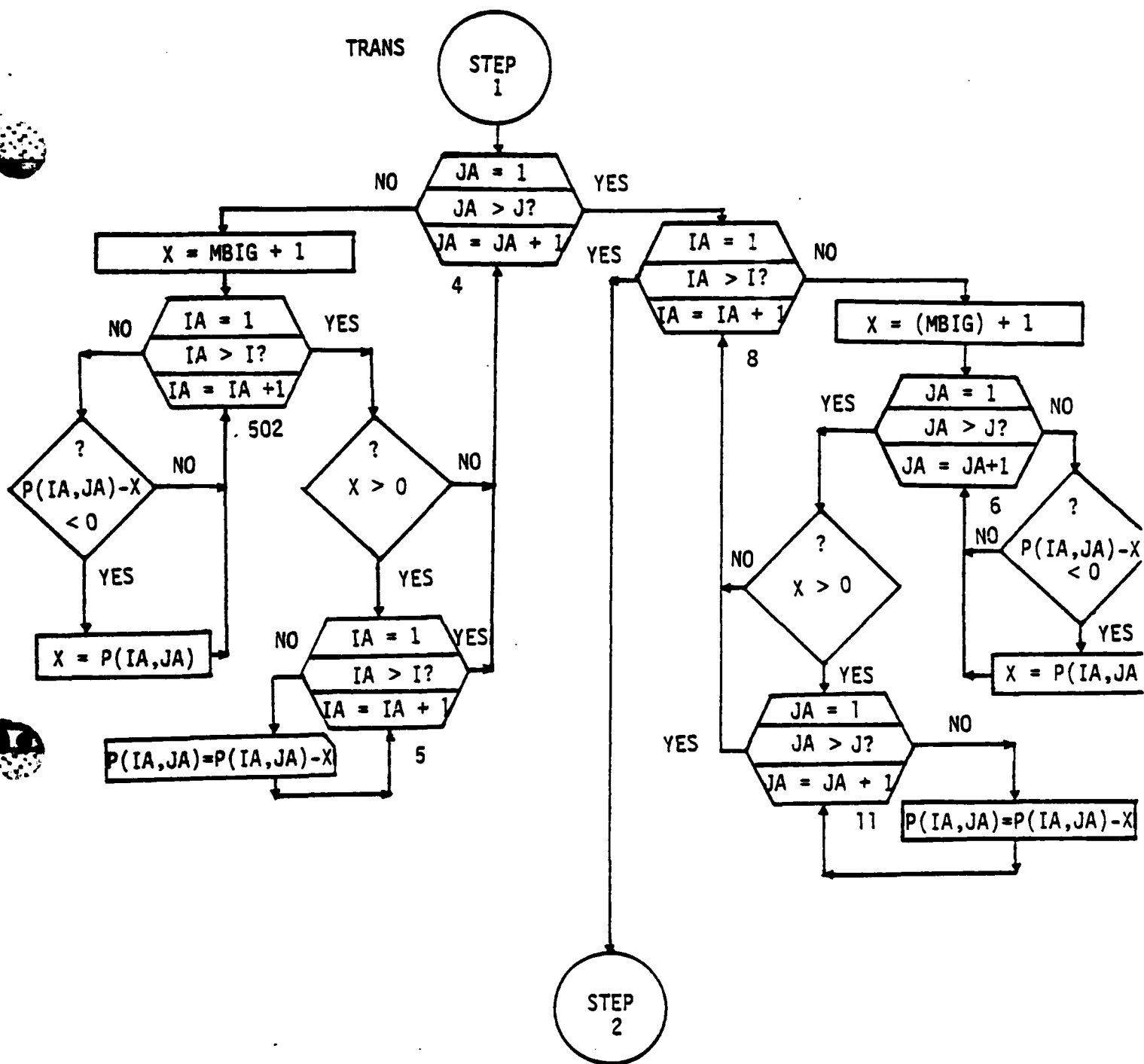


Figure 3-6. Subroutine TRANS (Continued)

TRANS

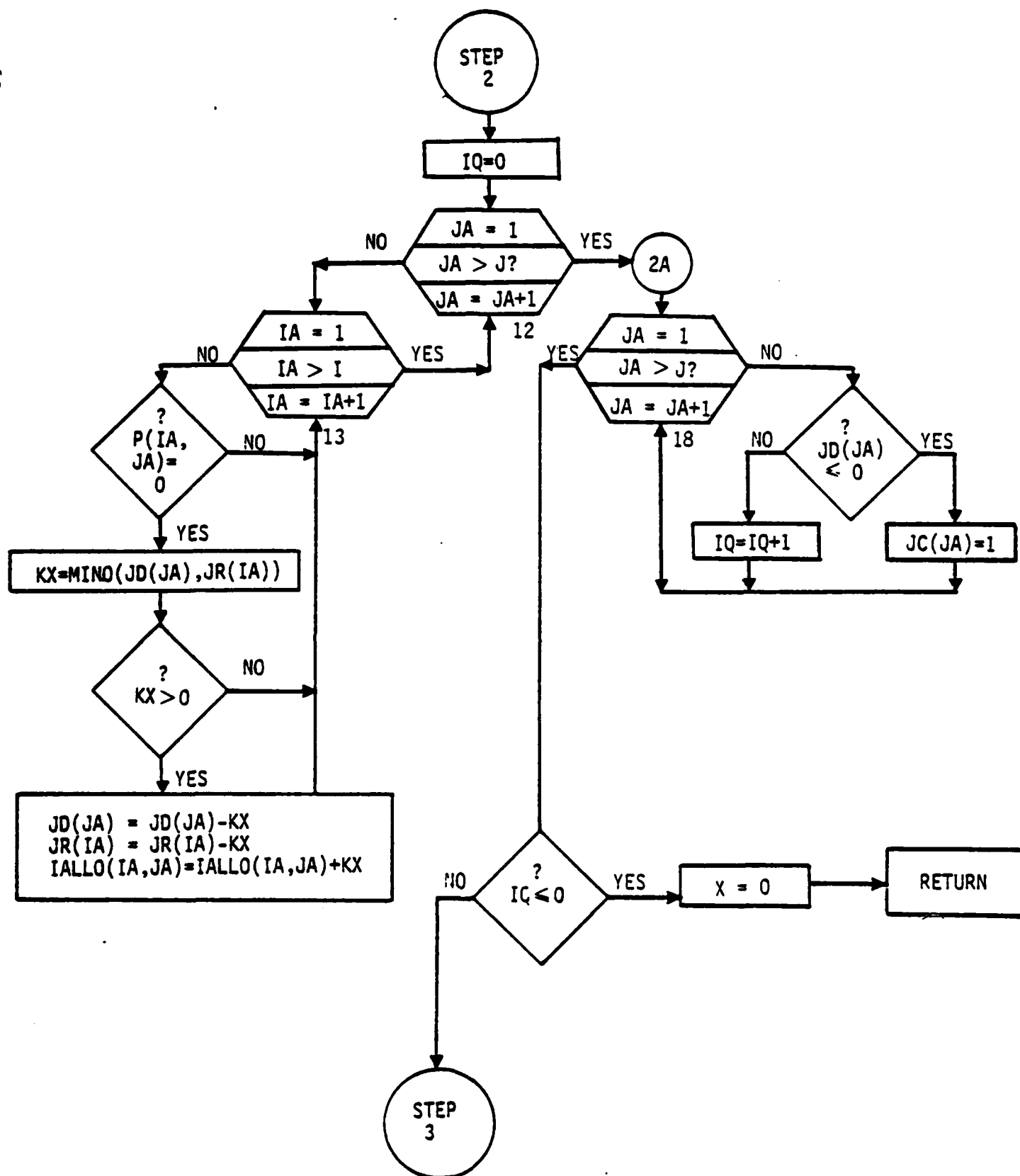
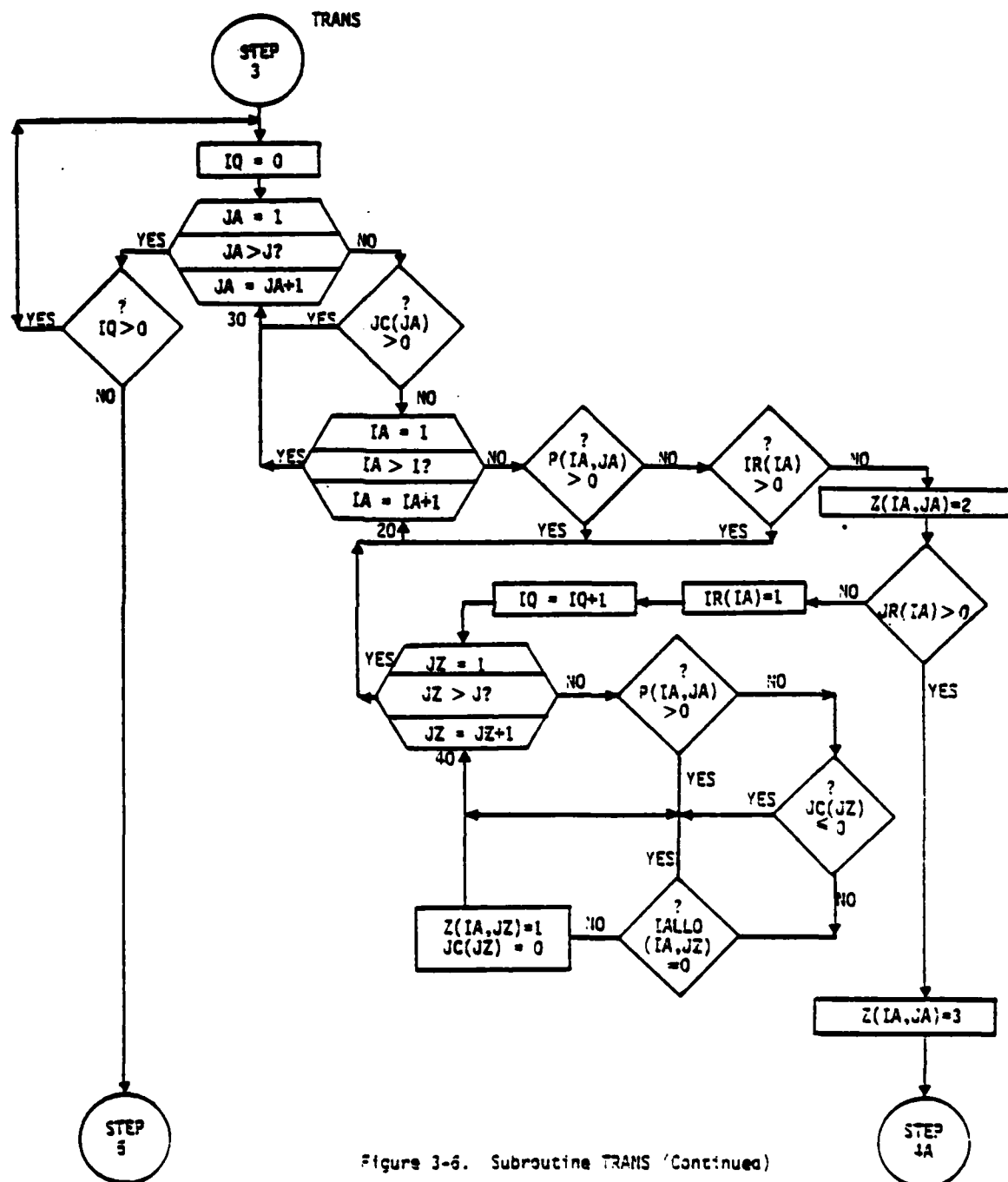


Figure 3-6. Subroutine TRANS (Continued)



TRANS

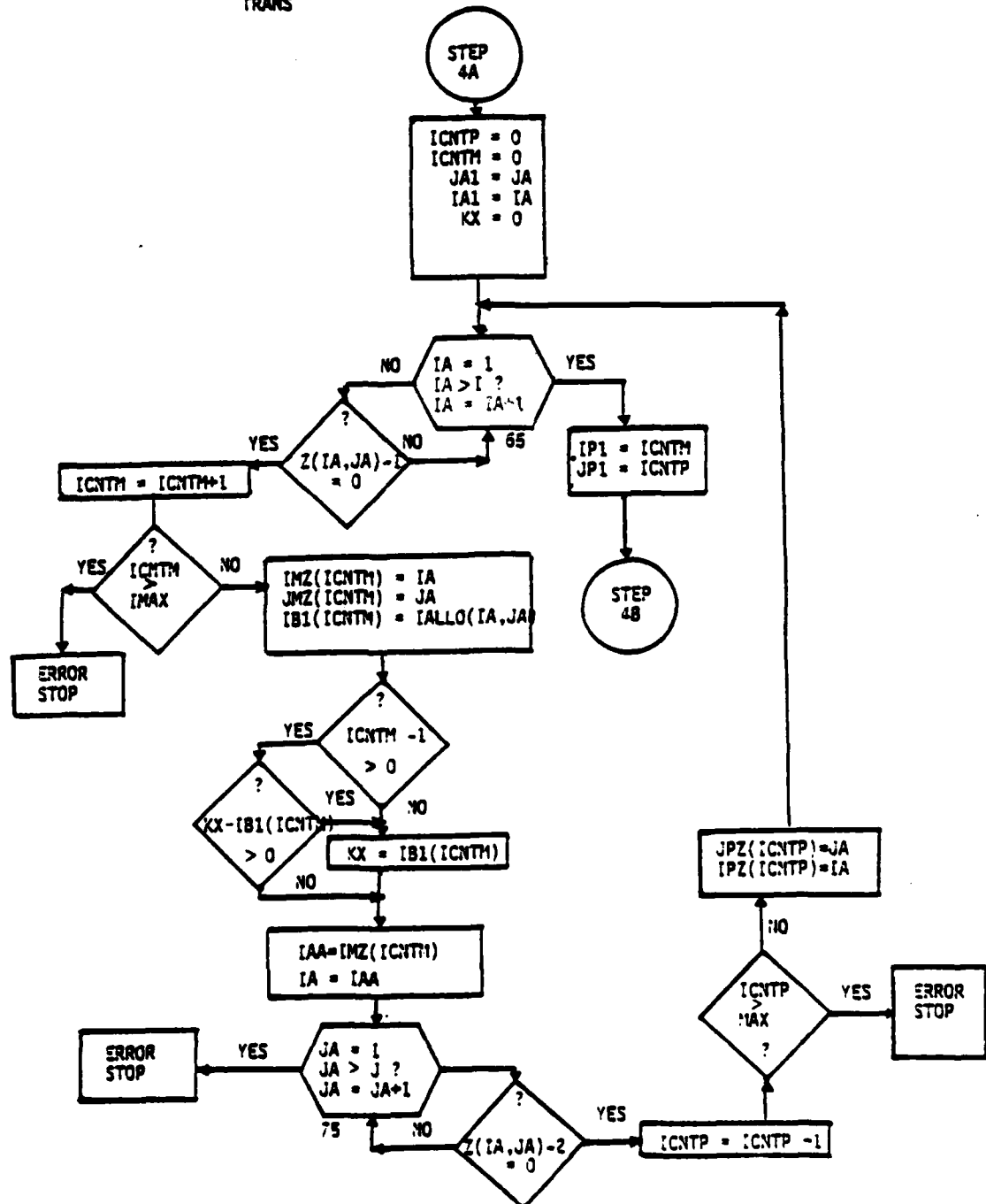


Figure 3-6. Subroutine TRANS (Continued)

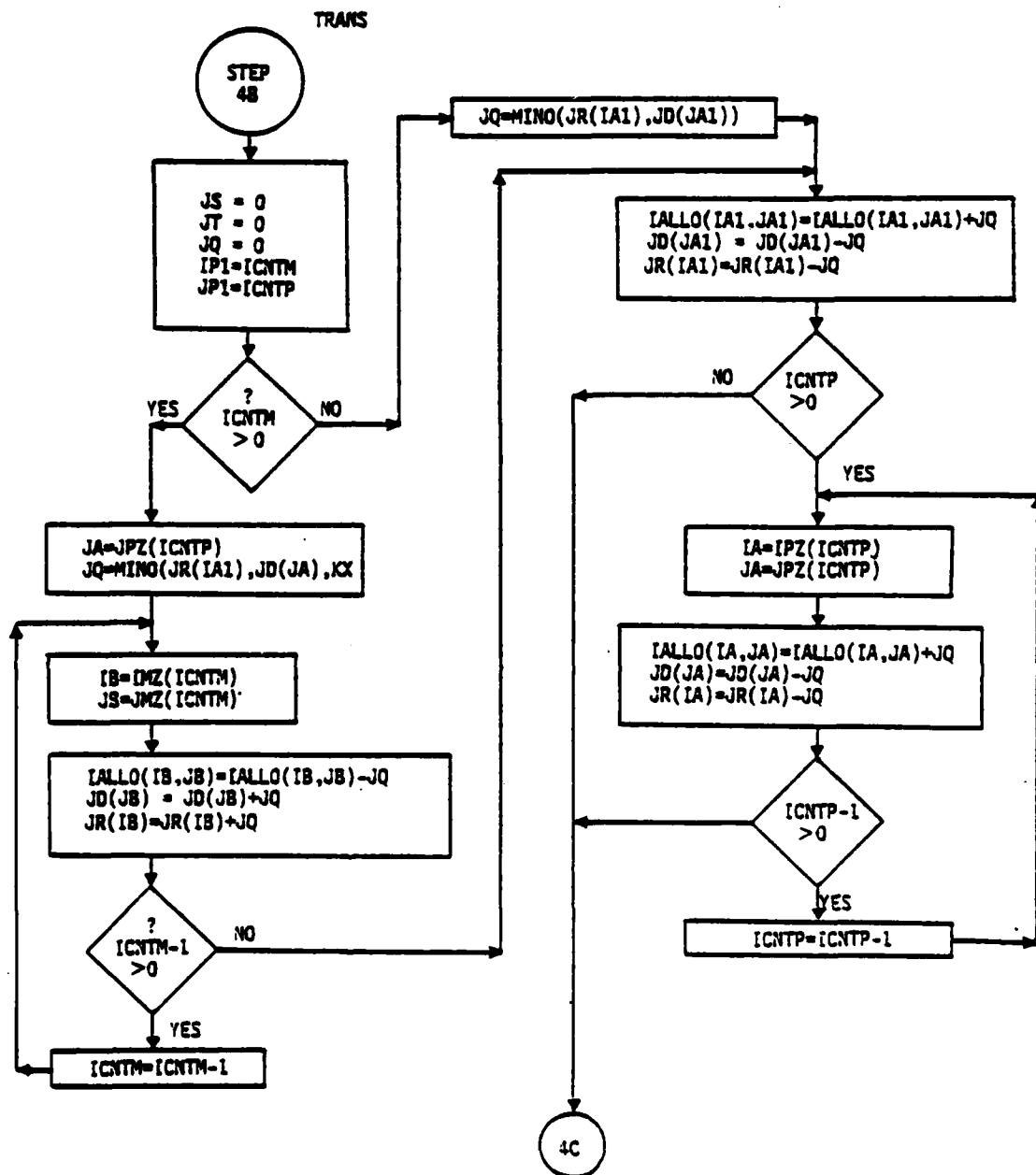


Figure 3-6. Subroutine TRANS (Continued)

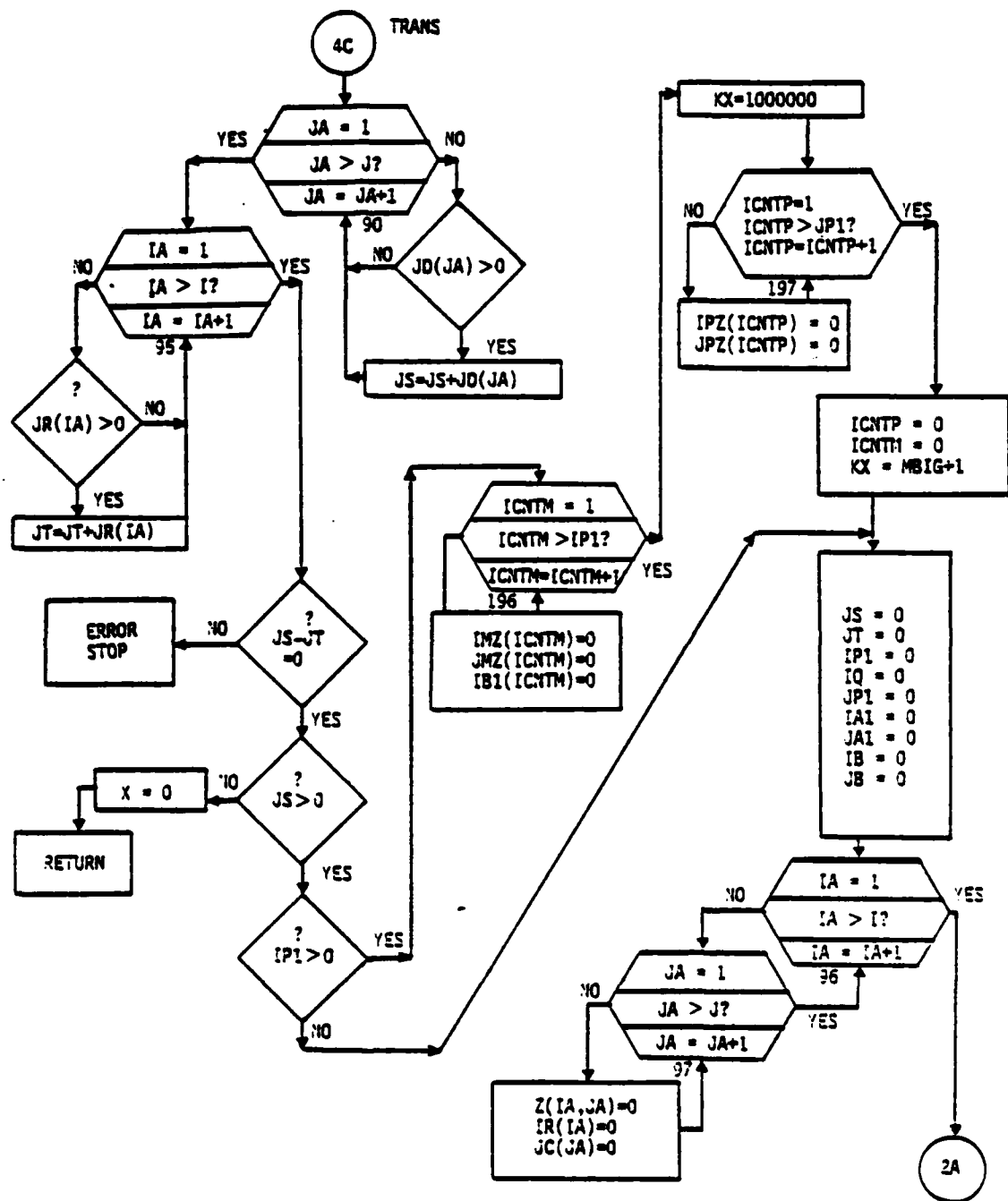
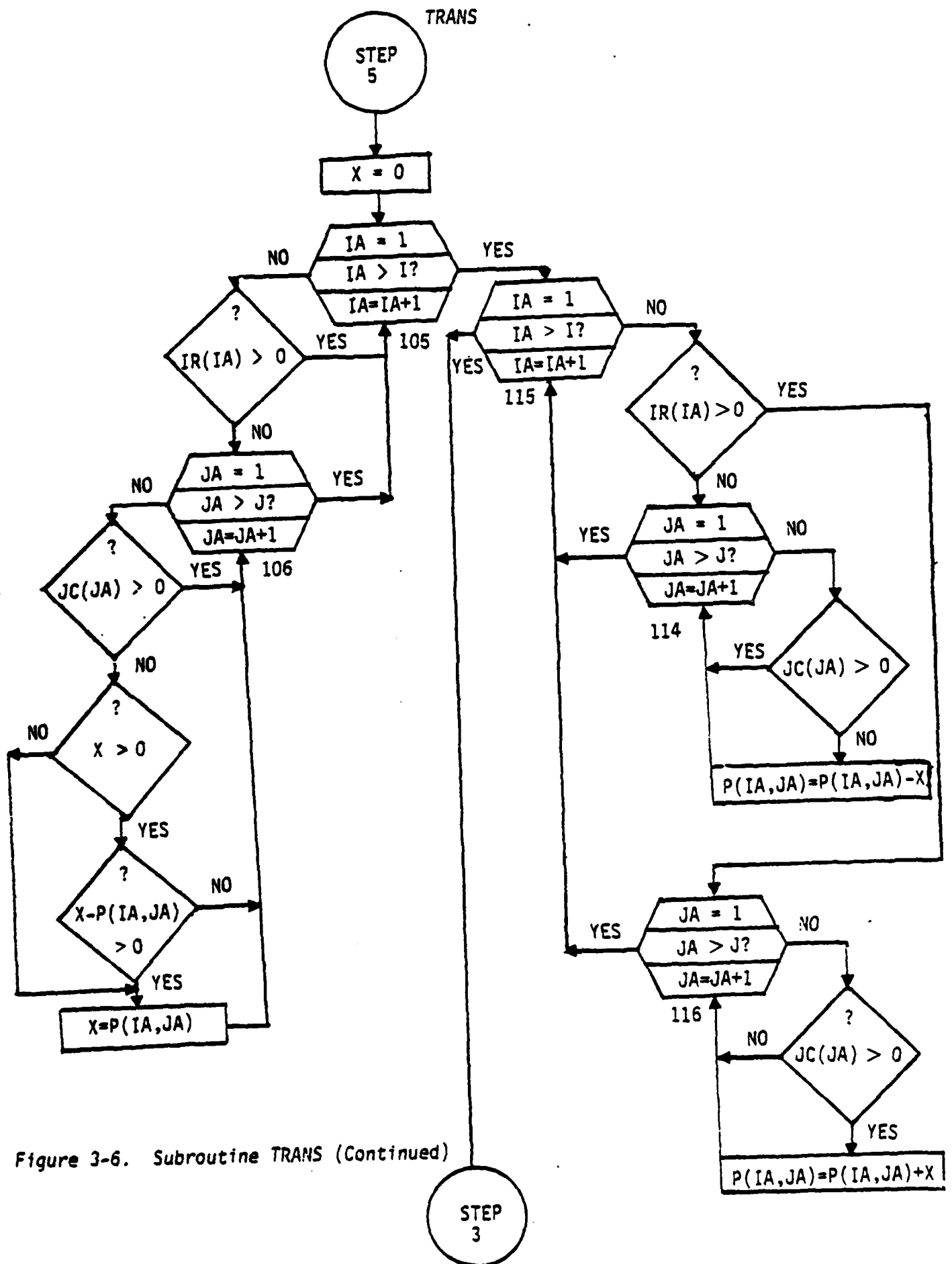


Figure 3-6. Subroutine TRANS (Continued)



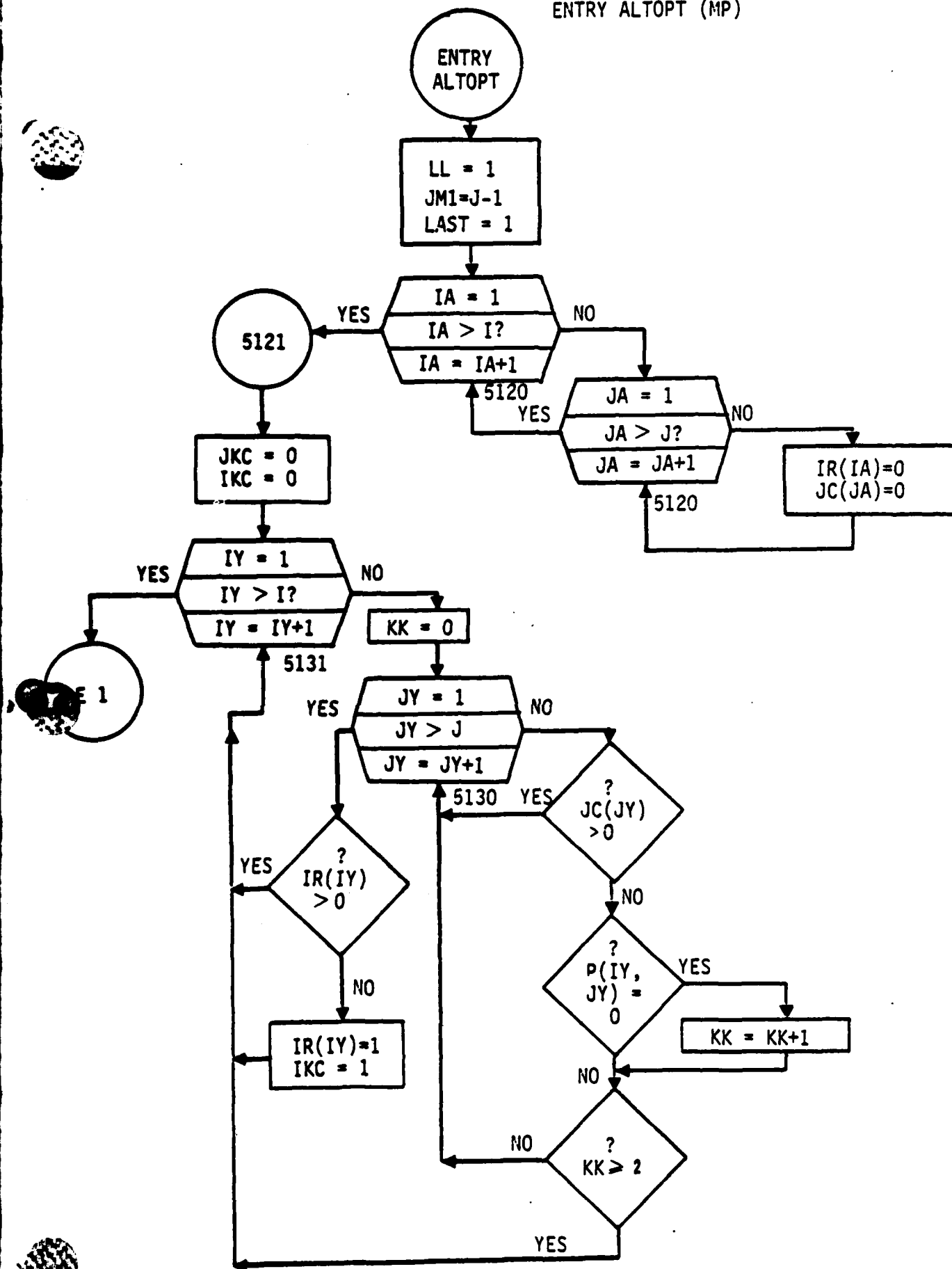


Figure 3-6. Subroutine TRANS (Continued)

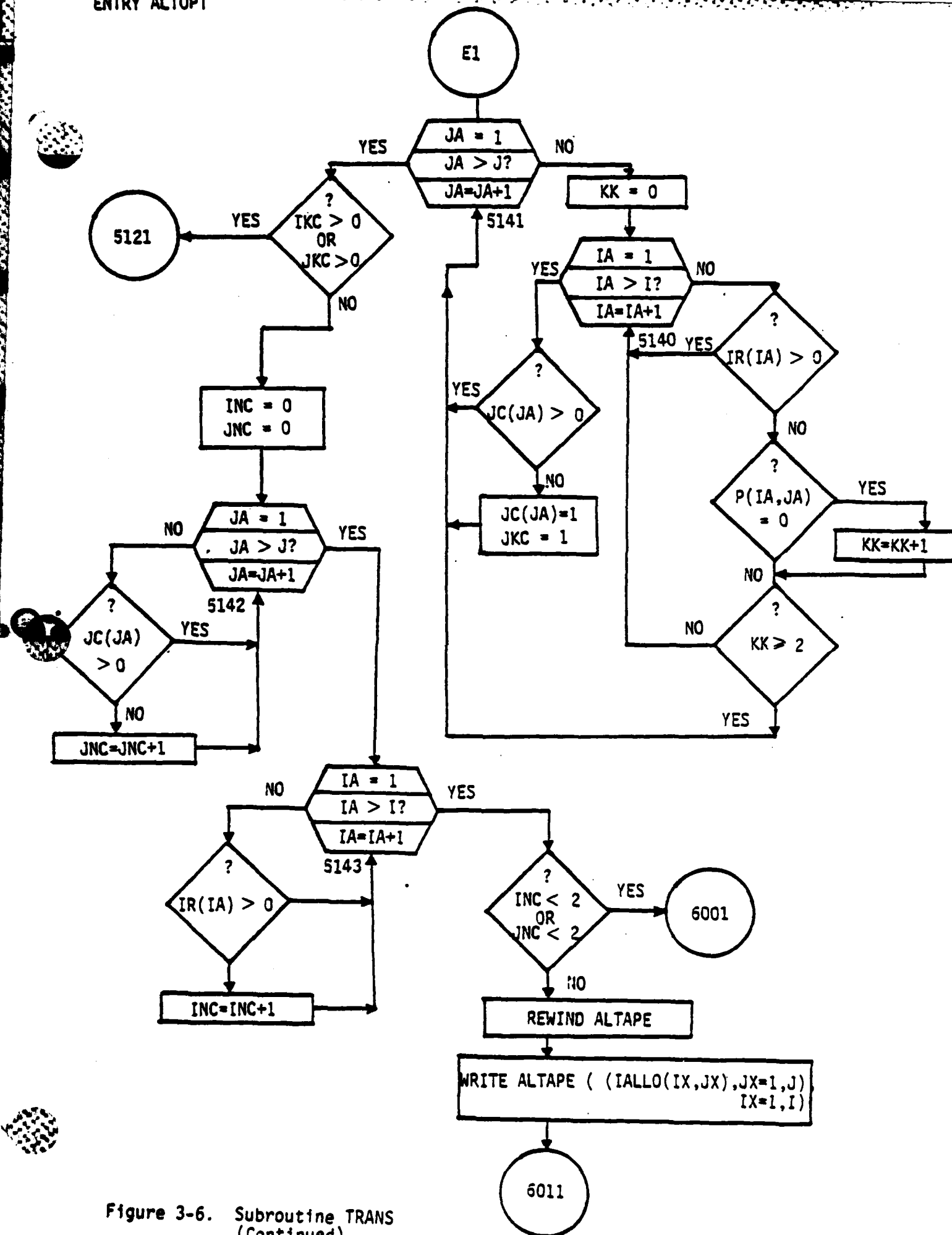


Figure 3-6. Subroutine TRANS
(Continued)

ENTRY ALTOPT

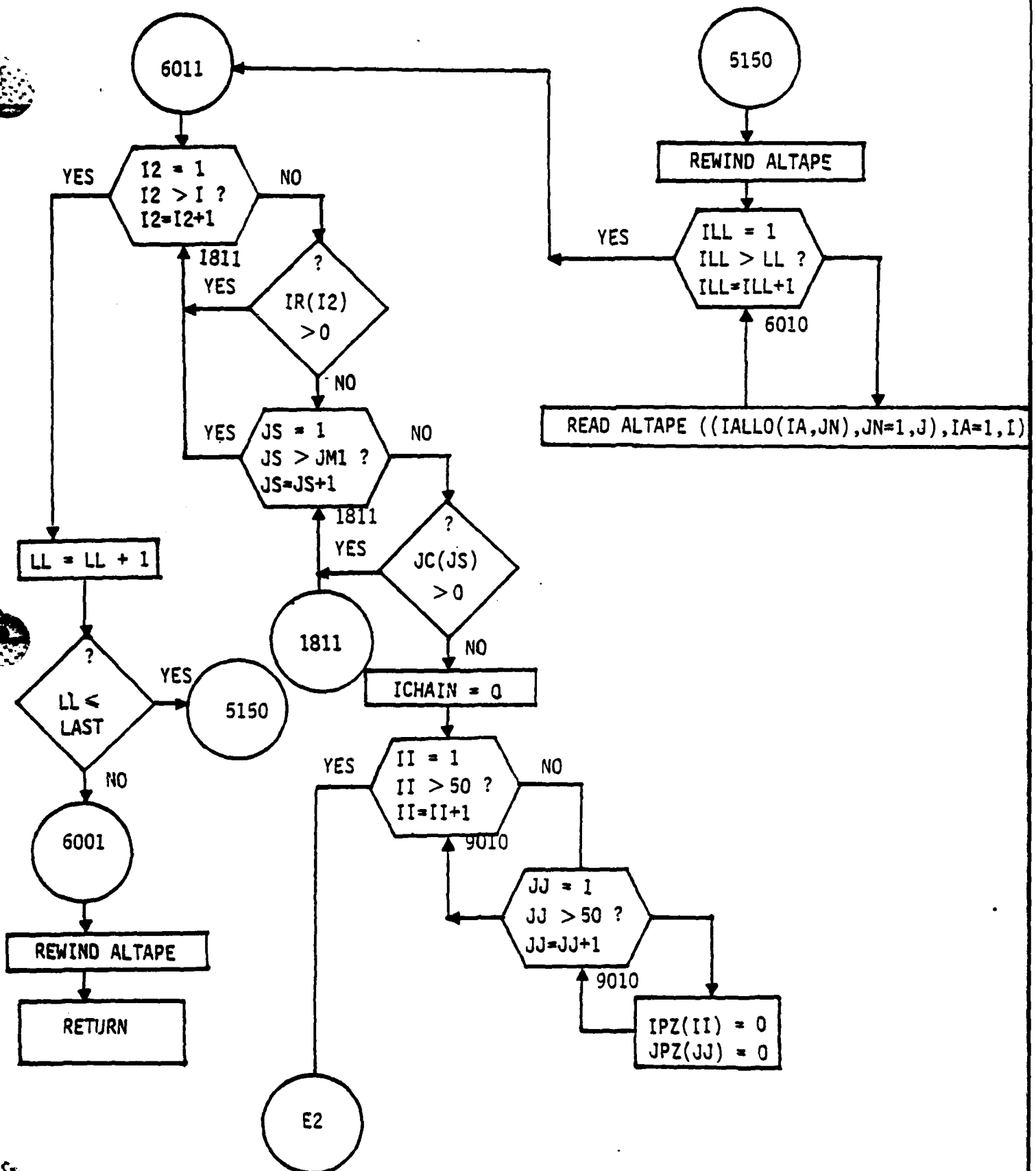


Figure 3-6. Subroutine TRANS (Continued)

E2

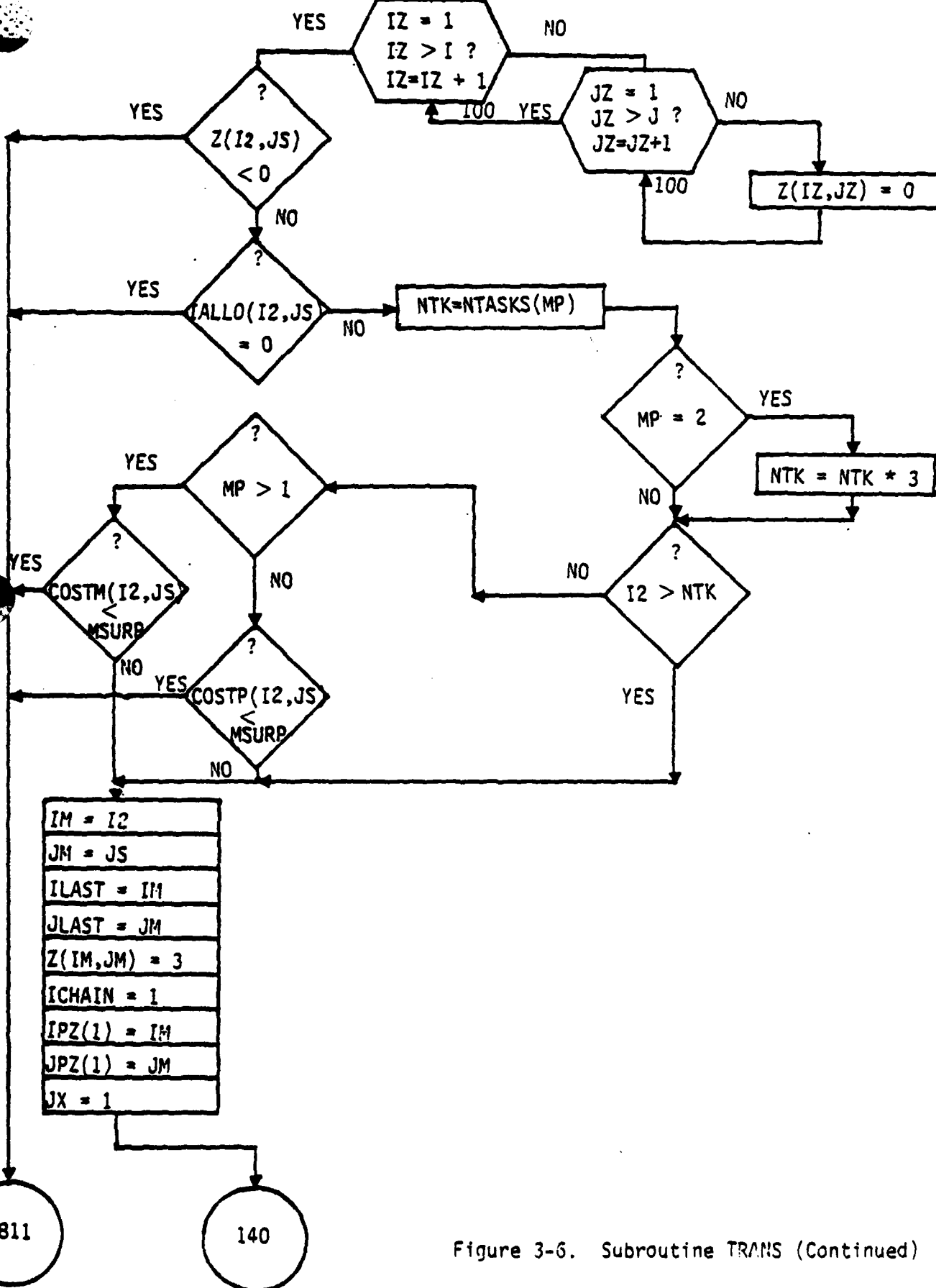


Figure 3-6. Subroutine TRANS (Continued)

ENTRY ALTOPT

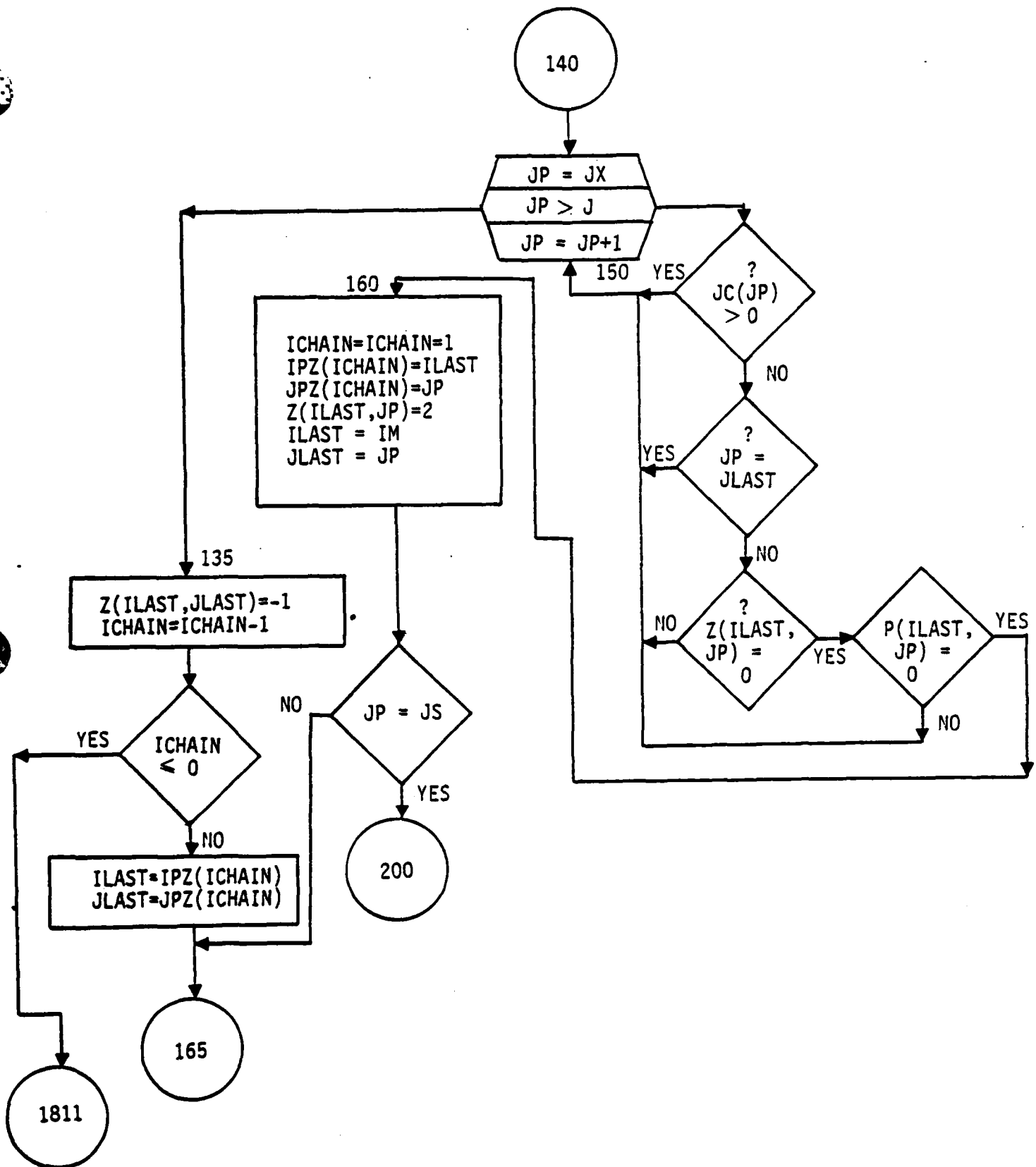


Figure 3-6. Subroutine TRANS (Continued)

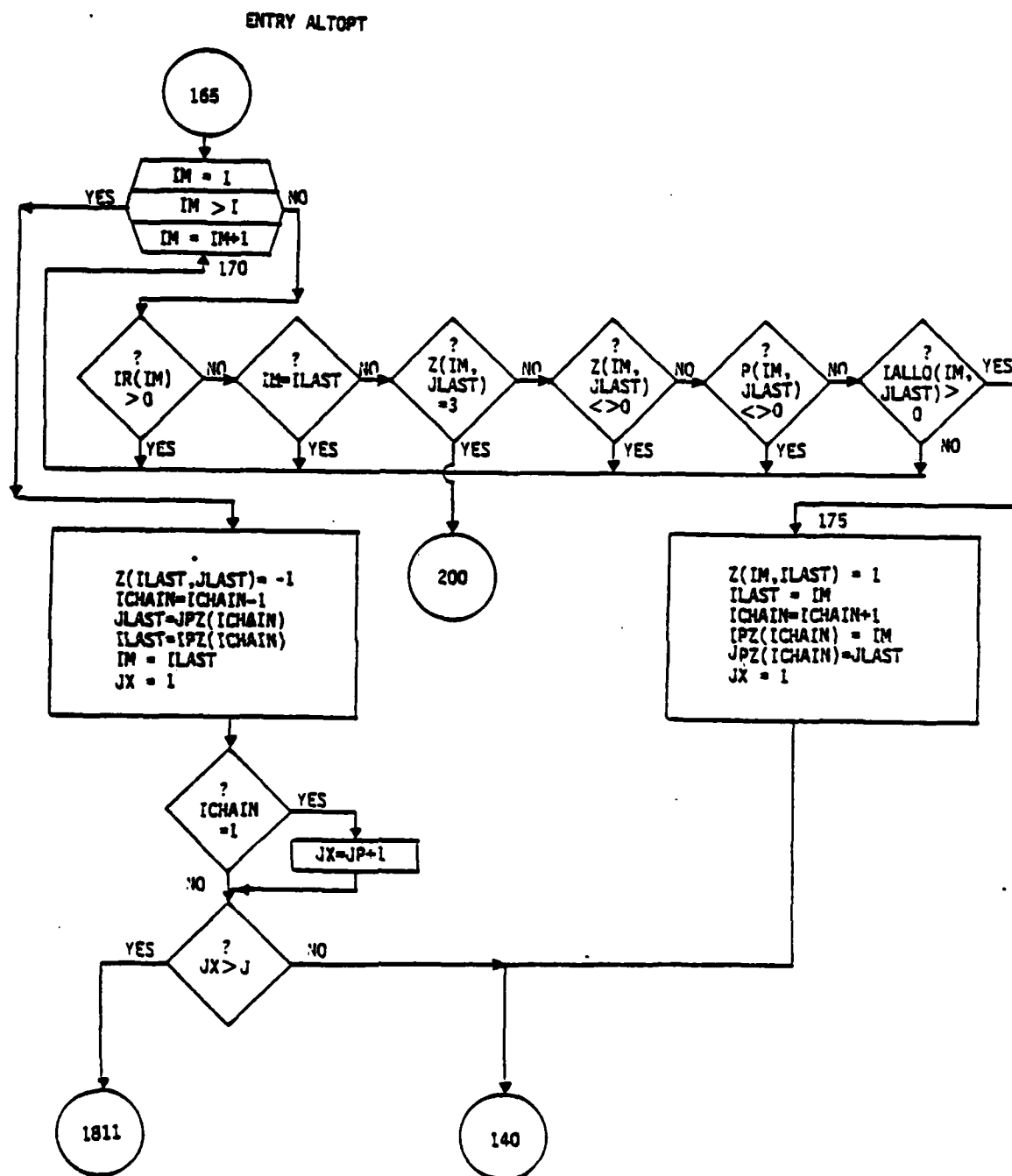


Figure 3-5. Subroutine TRANS (Continued)

ENTRY ALTOPT

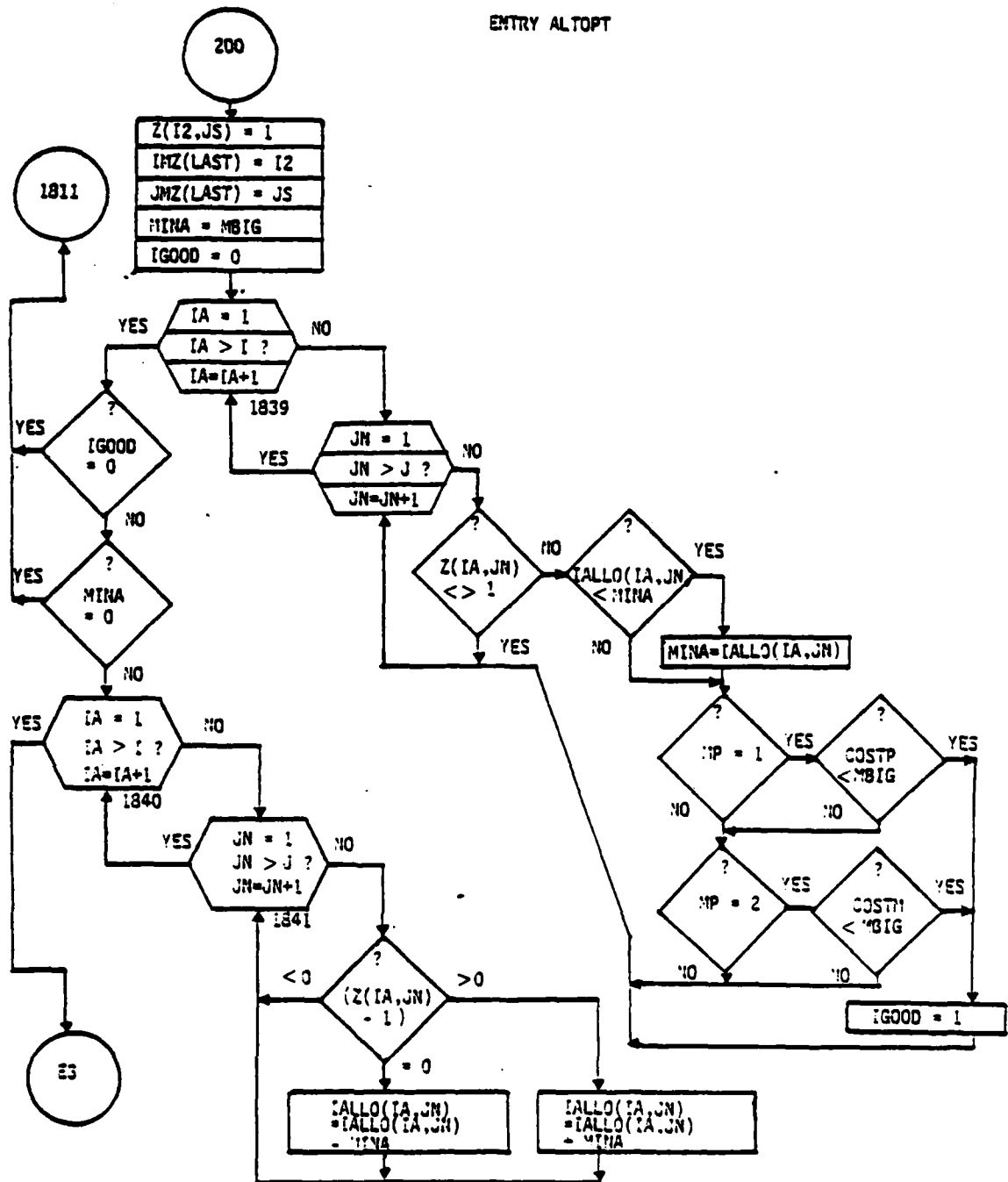


Figure 3-3. Subroutine TRANS (Continued)

ENTRY ALTOPT

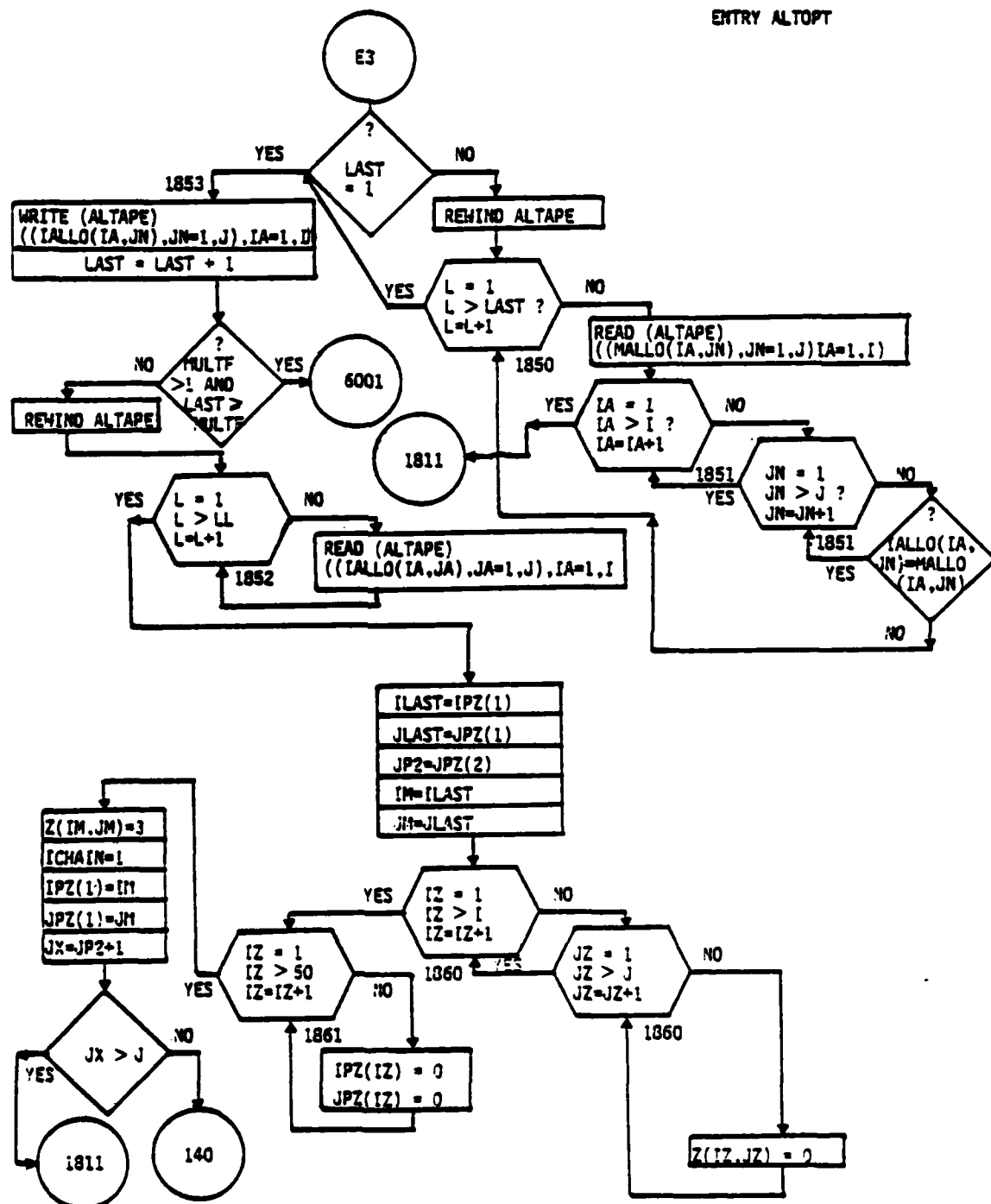


Figure 3-5. Subroutine TRANS (Continued)

3.6.2 Functions

Subroutine TRANS begins by initializing the various arrays it will use. Separate loops are used for personnel and materiel, although both perform the same basic functions. The surviving assets (from array IS) are put into array JR by task number. The team requirements for team NUMTRY (array ITEAMP or ITEAMM) are put into array JD by task number. The total items or individuals represented in these two arrays are then accumulated to get the total number of resources, ISUM, and the total number of demands, JSUM. Next, the cost matrix (P) which will be used is taken from either COSTP or COSTM, as appropriate. An extra column is added to this matrix and a cost of MSURP is put into every cell. This column provides the costs for assignments against any dummy demand, that is, surplus items.

A comparison of resource and demand determines the next step. If resources are greater than demand ($ISUM > JSUM$), a dummy demand equal to this difference must be created in order to complete the transportation problem. This, however, does not ensure a feasible solution. Developing feasible solutions, when only a complete feasible solution is required, would be very time consuming. In those cases the variable ILV is set to 1 by the calling routine and a task by task check is made to see if it is possible to fill the demand. If any demand is found impossible to fill the variable NOTEN is given a value of 1 and control is returned to the calling routine. If a possibility of fill exists for every demand, processing continues. If ILV was zero when the call was made this check is skipped entirely.

If the total demand was found to be greater than the total resources, a dummy supply is created to balance the two. The variable NOTEN represents the amount of dummy supply required. In this case no feasible solution is possible and if $ILV=1$ control is returned to the calling routine. If $ILV=0$ a complete solution is desired and

processing continues. A row is added to the cost (P) matrix and the cost in every cell is equal to MSURP, the cost for assignment of dummy resources.

The allocation array (IALLO) and the arrays IR, JC, and Z are zeroed prior to starting the transportation algorithm. The first step of the transportation algorithm is to modify the cost matrix (P).

Step 1 - The smallest value in each column is found and subtracted from each element in the column. The same procedure is followed for each row. This results in at least one zero in every row and every column.

Step 2 - Allocations are made through the zeros now in the cost matrix. The value of an allocation is the minimum of the row resources (JR) or the column demand (JD). This allocation is summed into array IALLO and subtracted from the resource (JR) and demand (JD) arrays.

Step 2A - After all possible allocations have been made a check of all demands is made. If the demand has been satisfied ($JD(JA)=0$), the column is flagged (covered) by a 1 in the appropriate element of array JC. If all demands are zero, a solution has been found and control returns to the calling routine. If unsatisfied demands remain, processing goes to Step 3.

Step 3 - Find an uncovered zero in the P array. This is a zero whose column flag (cover) in array JC is equal to zero and whose row flag (cover) in array IR is also equal to zero. The Z array is used for flagging the zeros found in the P array. Flags in the Z array then indicate possibilities for optimal allocations of resources. The search starts by finding uncovered columns, $JC()=0$. When an uncovered zero is found, it is flagged with a 2 in the Z array. If the

row of that $P=0$ has resources remaining its Z flag is changed to 3 and processing goes to Step 4. If the row does not have resources the row is flagged (covered) by a 1 in the appropriate element of array IR. This row is then checked for other elements of P which equal zero, lie in a column which has been previously covered, and through which a previous allocation has been made (the corresponding element of the IALLO array has a value greater than zero). This cell is then flagged with a 1 in the Z array and its column cover, JC(), is zeroed. Step 3 is repeated until an uncovered zero ($P=0$) is located with remaining row resources, thus directing processing to Step 4; or until all $P=0$ elements are covered, either row or column or both, and processing is directed to Step 5.

Step 4 - An uncovered zero element in the P array has been found with resources remaining. (If resources remain there must be a demand somewhere.) If the column of this zero element has a demand the allocation is made. If the demand of this column has been previously satisfied there will be a zero in this column which is flagged 1 in the Z array. Further, the row of that zero will contain a $P=0$ element with an associated Z flag equal to 2. (These elements are referred to as plus ($Z=2$) and minus ($Z=1$) zeros by Munkres.) The columns and rows are searched alternatively for these plus and minus zeros and their locations are stored by count in the arrays IMZ (ICNTM), row indices, and JMZ (ICNTM), column indices for minus zeros; and in arrays IPZ(ICNTP), row indices and JPZ(ICNTP), column indices of the plus zeros. Any prior allocations to the minus zero cells are stored in array IB1, again by count. This results in a chain of plus and minus zeros which guide an optimal reallocation. This chain will always have an odd number of elements and the last will always be a $P=0$ element flagged 2 (plus) in the Z array. The minimum value is found of the: resources in the row of the $Z=3$ element, the demand of the last plus zero, or the previous allocations made through the minus zeros. This is the maximum possible which can be reallocated, variable JQ. The reallocation is accomplished by subtracting this amount from previous allocations at the minus zero cells and adding to the allocations

of the plus zero cells. The associated resource and demand elements are adjusted accordingly. After the reallocation has been made the remaining resources and demands are each summed and checked against each other. If these totals are not equal an error STOP is made. If demands are equal to zero, a solution has been found and control is returned to the calling routine. If demands remain unsatisfied all flags and cover arrays are zeroed and processing returns to Step 2A.

Step 5 - Step 5 is entered from Step 3 when all zeros in the P array are found to be covered. The P array is searched to find the minimum uncovered ($IR()=0$ and $JC()=0$) element. This value is then subtracted from all uncovered elements in the P array and added to all twice covered ($IR()=1$ and $JC()=1$) elements. Processing then returns to Step 3.

Entry ALTOPT - The purpose of this section of subroutine TRANS is to locate any alternate optimal solutions. Possibilities for alternate allocations are found in a manner quite similar to the reallocation process in Step 4, above. An effort has been made in ALTOPT to reduce the scope of the search required. In order to form the chain of plus and minus zeros there must be at least two zeros in any row or column that is included in the solution. The first step in ALTOPT is to examine each row and then each column and cover by a 1 in the appropriate element of arrays IR or JC those that do not have at least two zero elements in the P array. The uncovered rows and columns are then counted and if either is less than two, no alternate solution is possible and a return is made.

If the possibility of an alternate solution does exist the original solution, array IALLO, is written to the file ALTAPE for reference. The search is then started for an uncovered allocation with an original cost equal to or greater than MSURP. This qualification is used to identify "choke points." Alternate solutions involving

changes in these allocations are the primary interest and therefore the search is limited to them. This allocation is the start point for the reallocation chain. Its row and column indices are filed in arrays IPZ and JPZ and the associated element in array Z is set to 3.

Next, the row of this allocation is searched for an uncovered zero element of array P. The Z array is checked for a zero value to ensure that this cell is not already included in the chain. The location is stored and Z is set at 2. If this is the column of the start element the chain is complete, otherwise proceed to the next step. If no unused $P=0$ is found, the Z of the previous element is set at -1 to indicate that no chain can be formed from it. The chain count is decreased to provide the location of the previous element in the chain. If this decreases the chain count to zero the process is started over.

The next step is to search the column of the last $P=0$ for an uncovered ($IR()=0$) allocation with P and Z elements also equal zero. If none is found the chain count is decreased, the Z of the previous element is set to -1, and processing goes to the row search or back to start. When an allocation is found the location is stored, the Z of this element is set to 1, and processing returns to the previous step to search the row of this element for a $P=0$.

The chain is complete when a zero P element in the column of the original $Z=3$ allocation is found. The Z for this cell is then changed from 3 to 1. The minimum allocation to the cells flagged by $Z=1$ is determined and the reallocation of this amount is made. A check is also made of the cost of each of these allocations, if the cost of every allocation is MBIG then only the fake assignment of existing assets is being rearranged and the solution is of no interest. Reallocation is made by subtracting this value from allocations previously made to those cells designated by $Z=1$ and by adding to those cells designated $Z=2$.

The next step is to determine if this is a unique alternate solution or a duplicate of a previously found solution. The variable LAST is used to count solutions and is initialized to 1 on entry to ALTOPT. If LAST=1 after a reallocation has been made, the solution must be unique and is simply written to the file ALTAPE. In all other cases the solution is compared, allocation for allocation, with each previous solution in the file ALTAPE. If all allocations are equal to any other solution this solution is discarded and search process is started over. If the solution is unique it is written to ALTAPE, LAST is incremented, and the variable MULTF is checked against the number of solutions found. If MULTF=1, the process continues until each solution found has been searched for alternates. If MULTF>1, the process stops when that number of solutions has been found or the possibilities have been exhausted. Each solution found is searched in turn for alternates. The variable LL is the indicator of which solution is to be read from the file ALTAPE to start each search.

3.6.3 COMMON BLOCKS

DLY3
GENEAL
KTR1
KTR2
PD2
PRNTIT
WK1
WK2

3.7 SUBROUTINE CAPT (MP, MF, NUMTRY)

3.7.1 General

Subroutine CAPT (Figure 3-7) provides control for the calculation of the capability (fraction of teams found) at each time of interest. Calculations are made by ICAP (initial, zero time, capability), WHEN (time when transfers are complete), and RCAP (capability at each time after zero). Capability calculations are made at each

of the input times and at zero time (before any transfers), at minimum time (immediately after transfers start), and at infinite time (all possible transfers made). The argument MP designates personnel (1) or materiel (2), MF identifies the mission number, and NUMTRY is the number of teams which can be formed as determined by MAXT.

CAPT initializes JMIN to zero and JMAX to the number of teams built plus one, prior to calling ICAP. If no teams could be constructed (NUMTRY=0) the initial as well as all other capabilities are zeroed and no subroutine calls are made. JMIN and JMAX are used by ICAP for a binary (two number) search to find the team which can be formed using survivors in their own job only. This number of teams is used to calculate TOTCAP (1, MF, MP), the zero time capability.

CAPT then calls subroutine WHEN for the calculation of the time when each transfer will be completed and ready for operation. The array RETURN is constructed by subroutine WHEN and contains, for each time point, the number of each task available for team completion.

The array RETURN is then used by RCAP where the team requirements are matched to the available assets at each time. The capability, TOTCAP, is calculated by RCAP for all times other than zero.

3.7.2 COMMON BLOCKS

GENERL
PD2
STATG
SURV

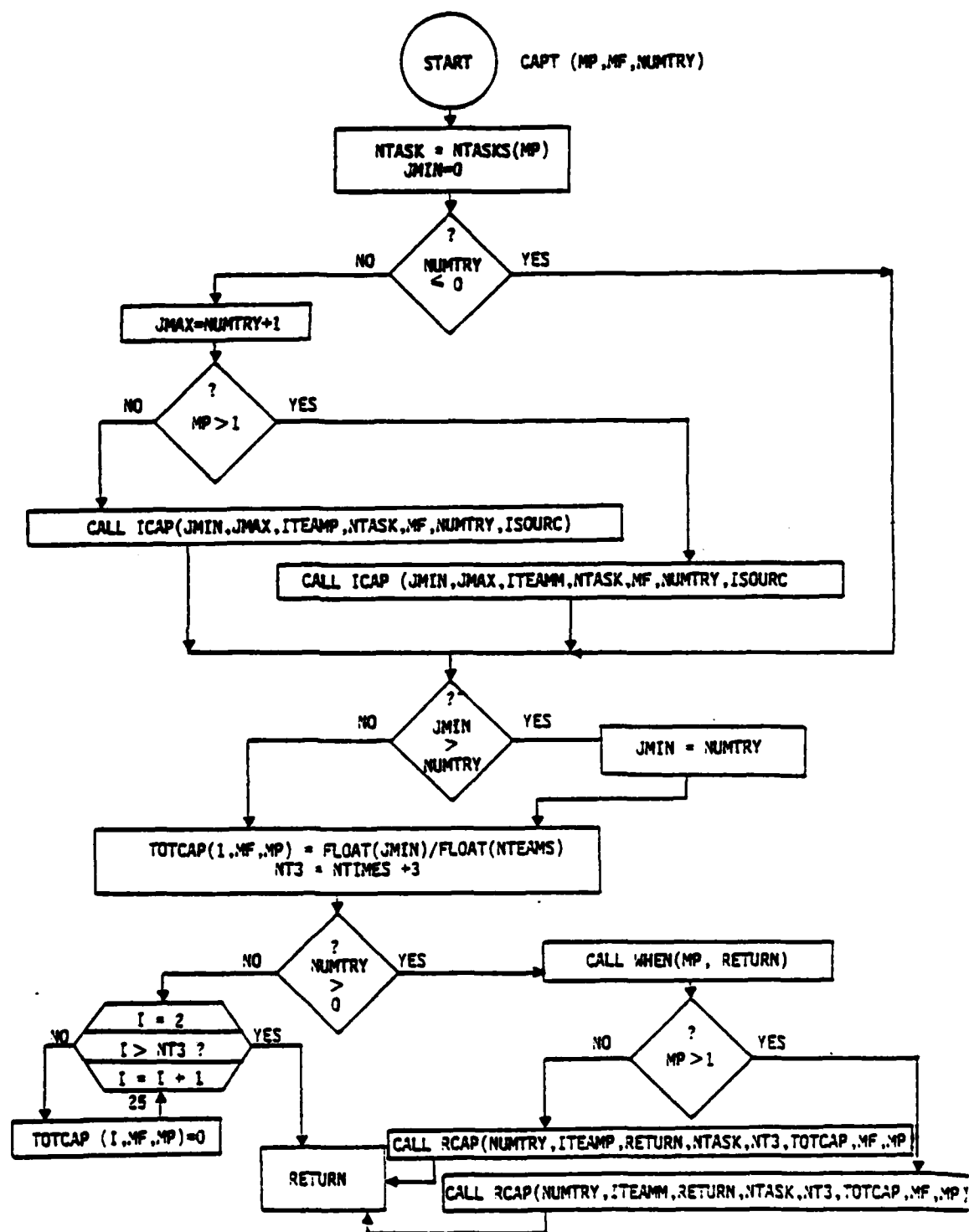


Figure 3-7. Subroutine CAPT

3.8 SUBROUTINE ICAP (JMIN, JMAX, ITEAM, NTASK, MF, MAX, IS)

3.8.1 General

Subroutine ICAP (Figure 3-8) is called by CAPT to find the team which can be constructed using survivors in their original job only, that is before any transfers have been considered. The arguments JMIN and JMAX provide the initial bounds for the binary search, of zero and one more than the maximum number of teams built (NUMTRY + 1). ITEAM is a dummy for the set of team requirements, ITEAMP or ITEAMM, being passed by CAPT. NTASK is the number of task lines being considered, MF is the mission number, MAX is the maximum team possible, and IS is the array of survivors for this iteration.

Subroutine ICAP compares the survivors in each task line with the requirements of a particular team number. If any task line has fewer survivors than the requirement, then that team cannot be formed. The first try is made using the team number mid-way between zero and NUMTRY+1. If a team cannot be completed then JMAX is set at that team number. When a team is successfully completed JMIN is set to that team number. In either case the process is repeated for the team number mid-way between JMIN and JMAX. The process stops if JMAX is reduced to one, JMIN is increased to MAX, or the difference between JMIN and JMAX is one. In any of these cases, except zero, JMIN is the number of the last successful team and is passed back to CAPT.

3.8.2 COMMON BLOCKS

Subroutine ICAP contains no common blocks.

ICAP(JMIN,JMAX,ITEAM,NTASK,MF,MAX,IS)

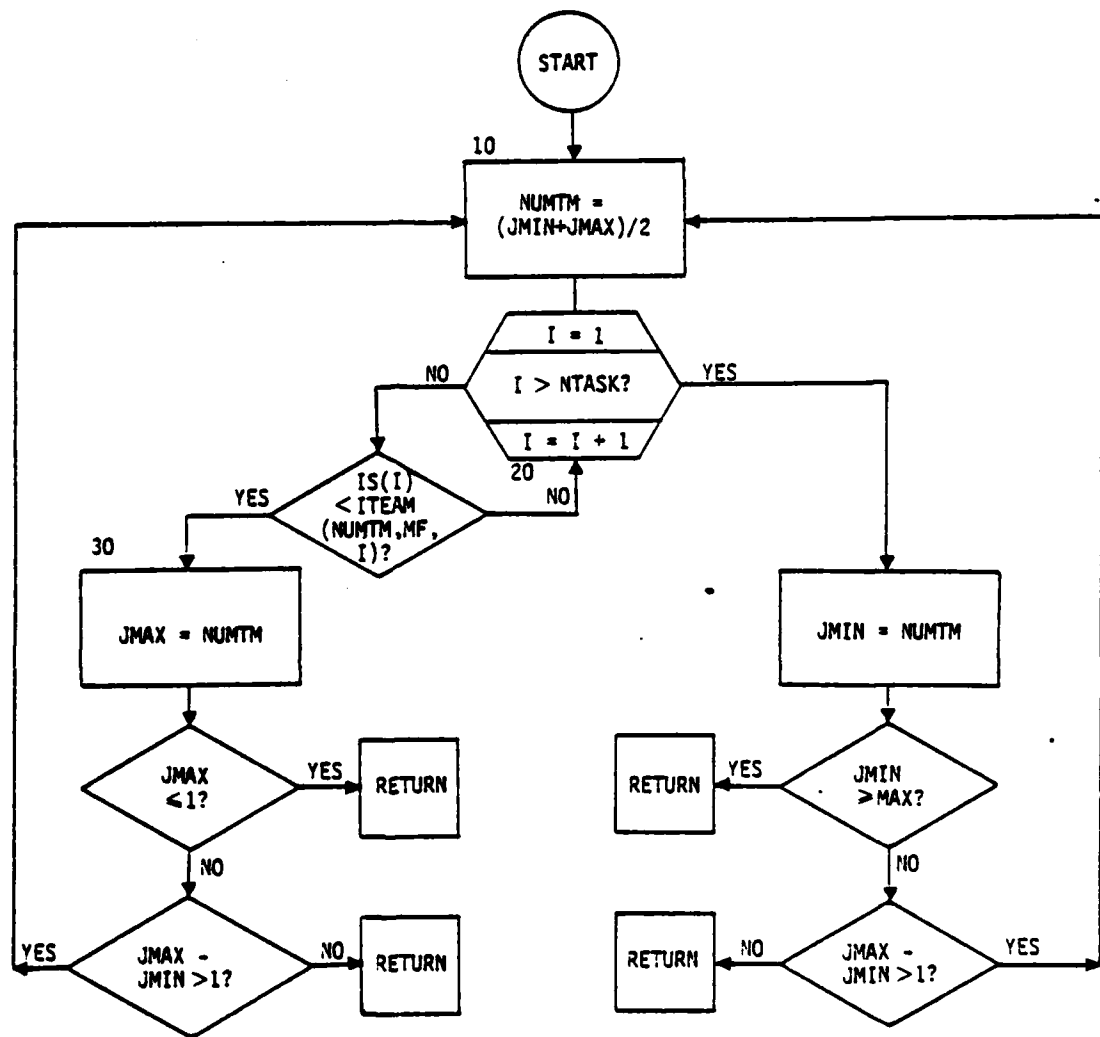


Figure 3-8. Subroutine ICAP

3.9 SUBROUTINE WHEN (MP, RTN)

3.9.1 General

Subroutine WHEN (Figure 3-9) determines the availability time for all assignments made by subroutine TRANS. The argument MP defines for the subroutine either personnel (1) or materiel (2). RTN is the array, to be constructed by the subroutine, which will contain the number of individuals or items available at each time for each task. The time of availability is determined by summing all the times associated with a particular transfer. These include transfer time, delay (commander's decision) time, and for materiel the repair time when appropriate. An option, IMEANT, is available to use the mean times as input (IMEANT=1) or to use an exponential distribution for each time (IMEANT=0). This distribution is calculated by drawing a random number, RANDOM, and getting a time, t , from the equation; $t = \bar{t} \ln(\text{RANDOM})$, where \bar{t} is the input mean time.

Subroutine WHEN contains separate loops for personnel and materiel which perform the same basic functions. The first step is to find all allocations with zero cost which have been made to their primary position (resource task number equals demand task number). These allocations are then put into the RTN array by task number for each time of interest, except zero time. (Zero time calculations are made by subroutine ICAP considering survivors in their primary duty, only.) These allocations are the only ones stored in array RTN (1, Task No.) and subsequently considered for the "minimum time" team build. The assumption is that even zero time transfers to another position result in some disruption of the team.

The second step is to again examine the diagonal elements of the allocation matrix, this time for those that have some cost associated. This step provides the capability to represent delays

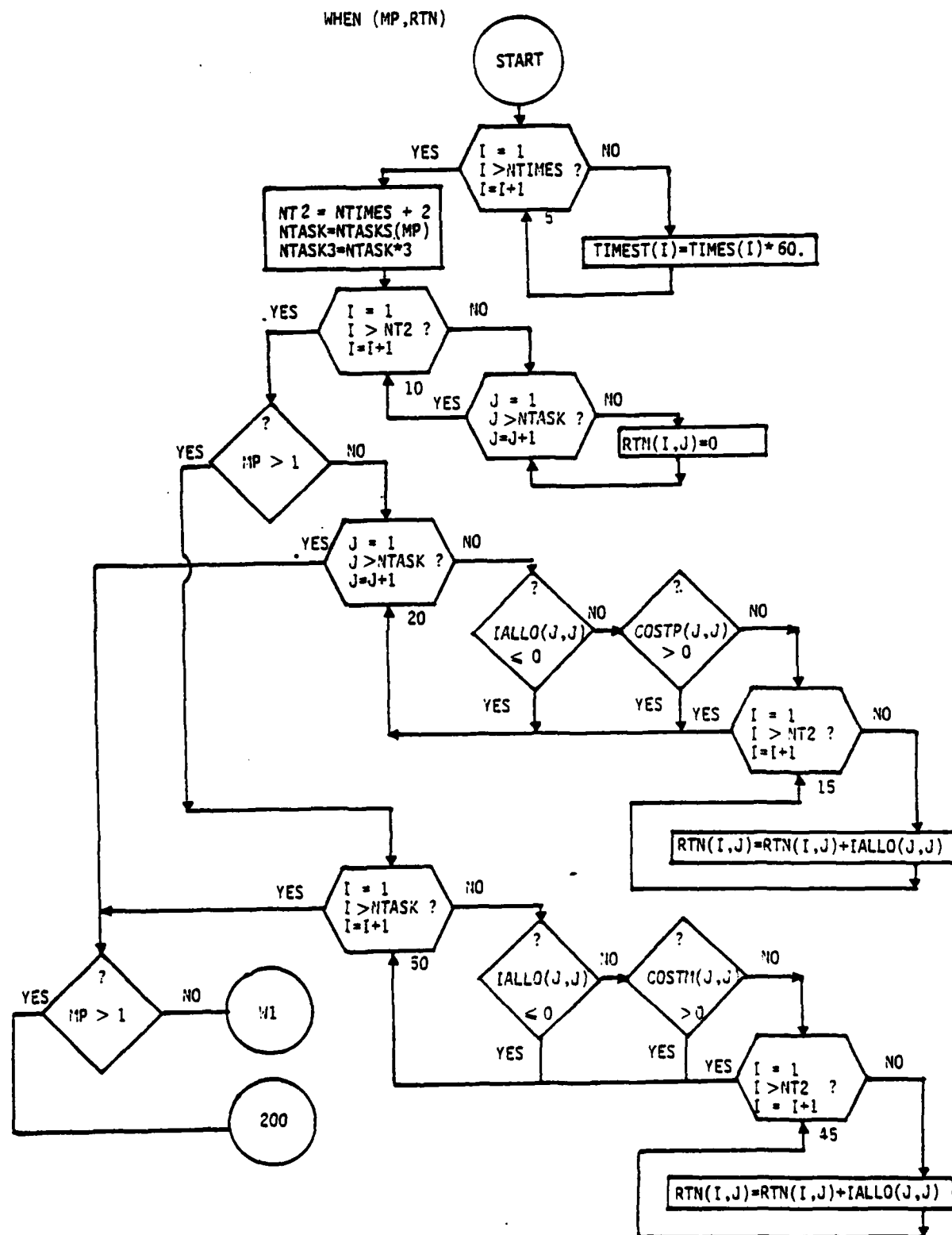


Figure 3-9. Subroutine WHEN

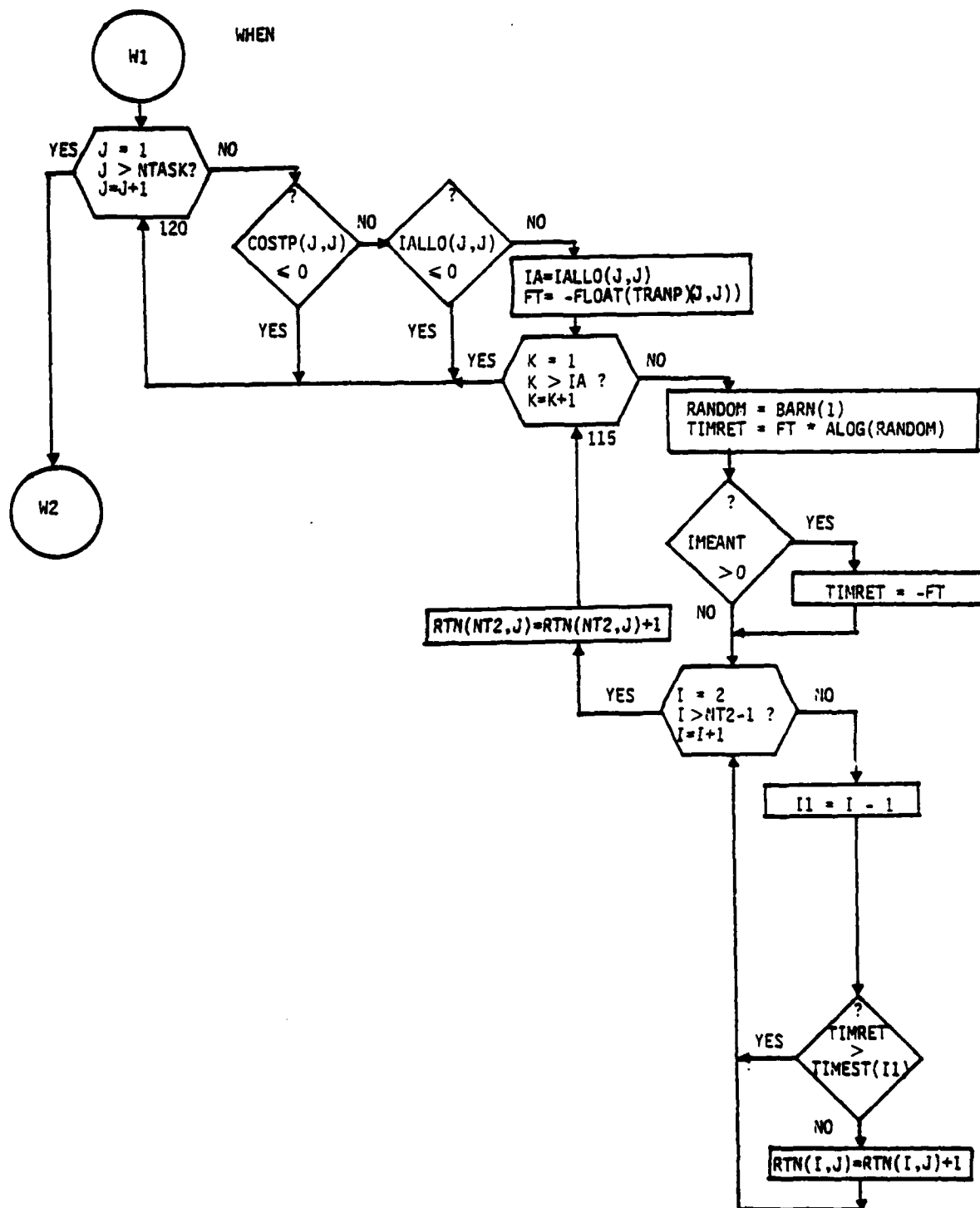


Figure 3-9. Subroutine WHEN (Continued)

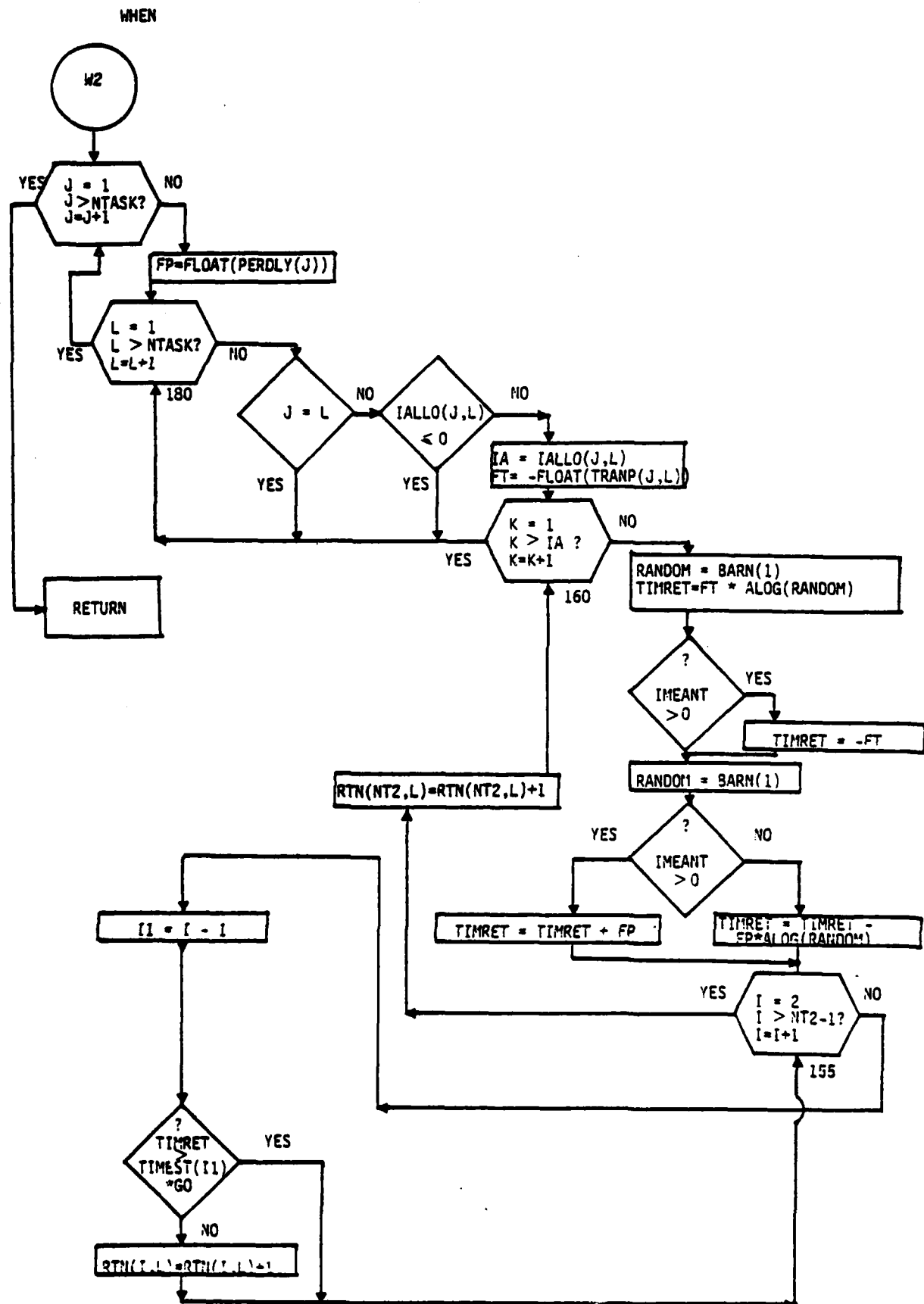
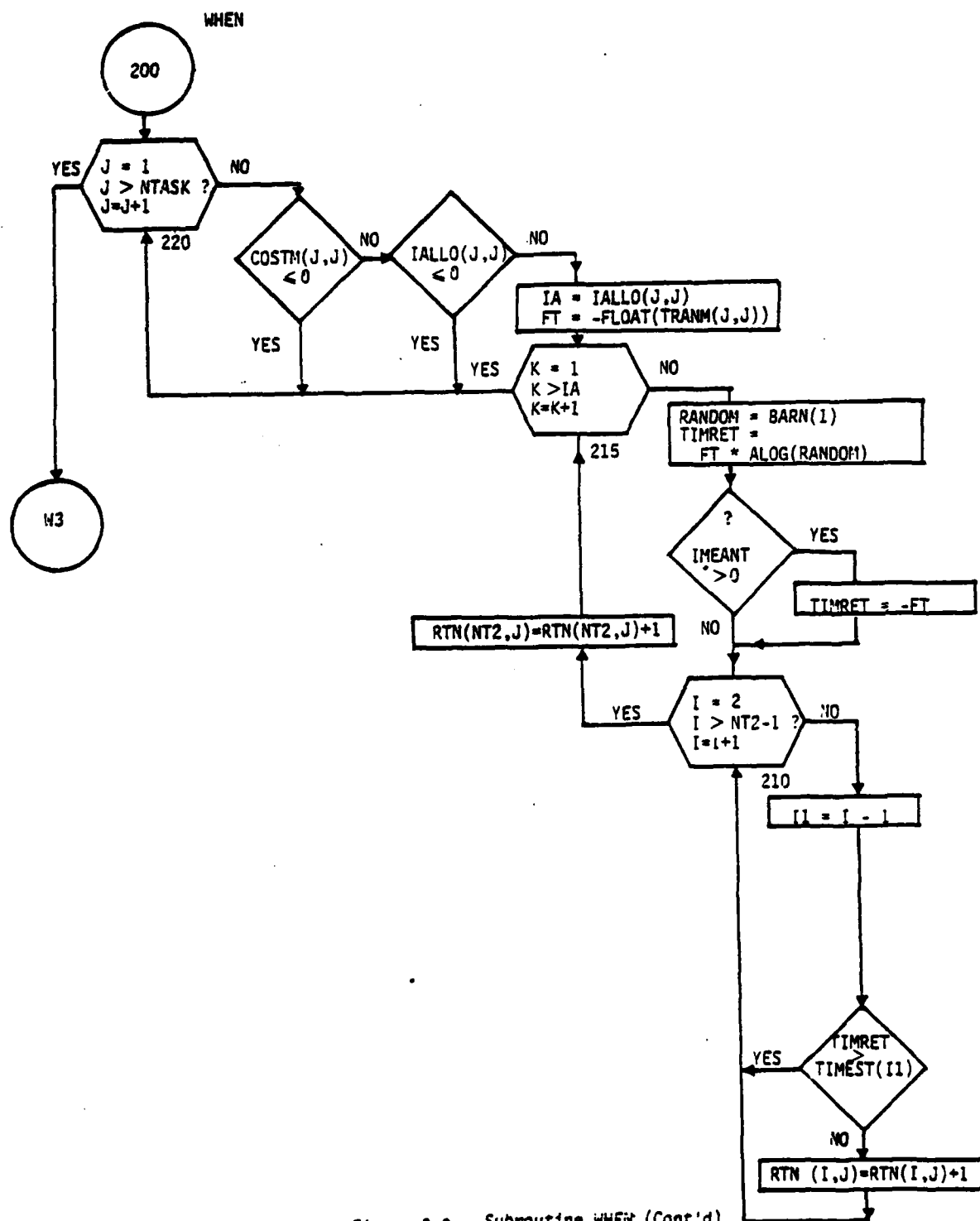


Figure 3-9. Subroutine WHEN (Continued)



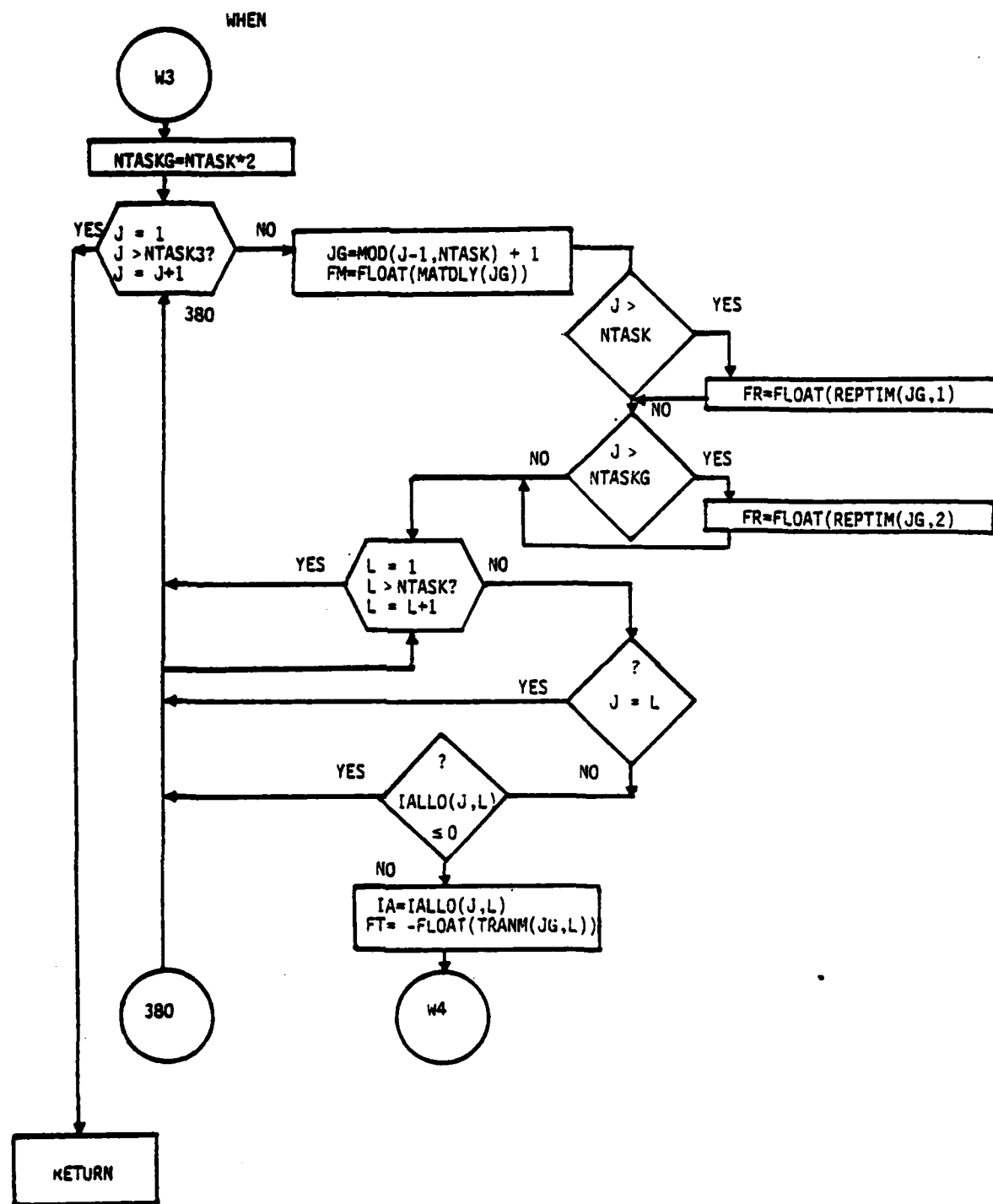


Figure 3-9. Subroutine WHEN (Continued)

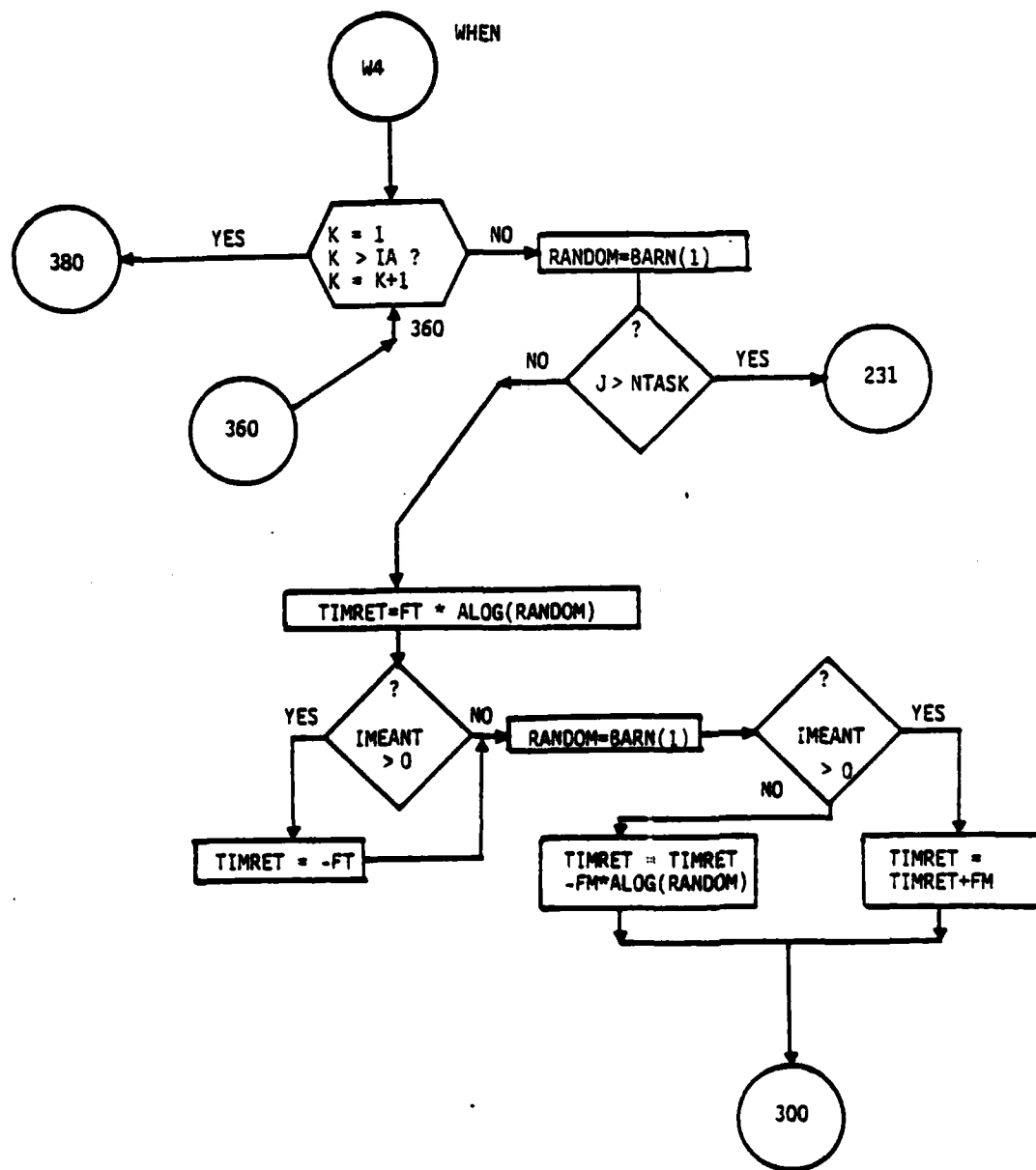


Figure 3-9. Subroutine WHEN (Continued)

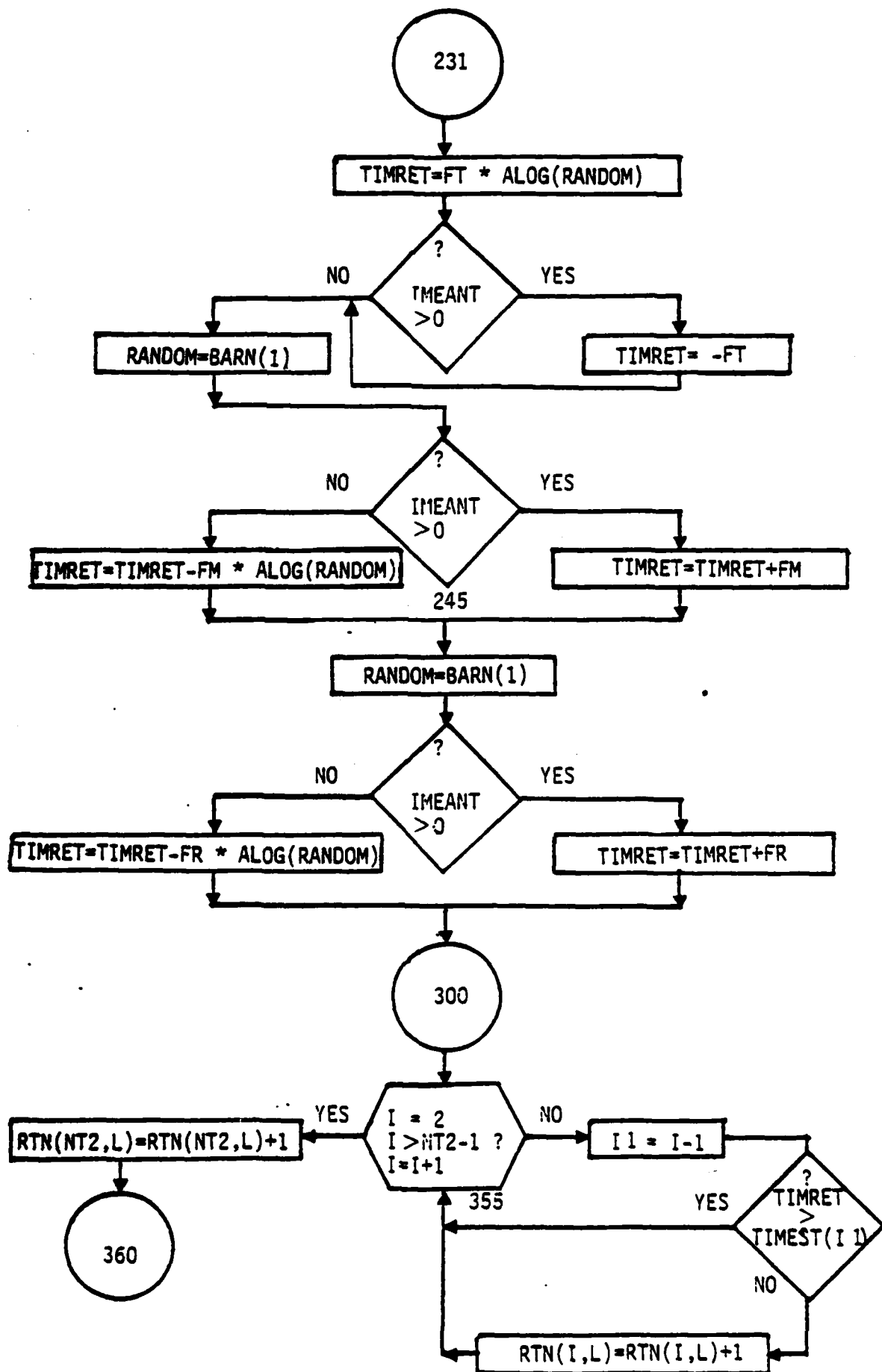


Figure 3-9. Subroutine WHEN (Continued)

for individuals or items filling their own jobs. This may be appropriate in some applications. The only time considered here is transfer time. Commander's decision delay time is not applied to any assignment to that item's primary function. The transfer time is either modified by the exponential or used directly. Times are drawn for each individual item of the allocation and each item is added to the RTN array for the appropriate, and all subsequent, times. If the return time, TIMRET, is greater than the greatest time of interest the item is added to the RTN array for "infinite time."

Finally, all other (non-diagonal) allocations are examined. The same steps as above are followed with all appropriate times being considered for each individual of an allocation. The resulting array RTN is returned to CAPT for subsequent use by subroutine RCAP.

3.9.2 COMMON BLOCKS

DLY1
DLY2
DLY3
GENERL
SEED
WK1

3.10 SUBROUTINE RCAP (MAX, ITEAM, RTN, NTASK, NT3, TOT, MF, MP)

3.10.1 General

Subroutine RCAP (Figure 3-10) calculates the capability, fraction of total teams formed, for personnel or materiel at each of the times of interest, except zero time. (Capability at zero time is calculated by subroutine ICAP.) This is accomplished by comparing the available assets at each time, given in array RTN, to the requirements for the teams. When a requirement is found that exceeds the available assets, that team cannot be completed. The previous team number is used to calculate the fraction of total teams available at

RCAP(MAX,ITEAM,RTN,NTASK,NT3,TOT,MF,MP)

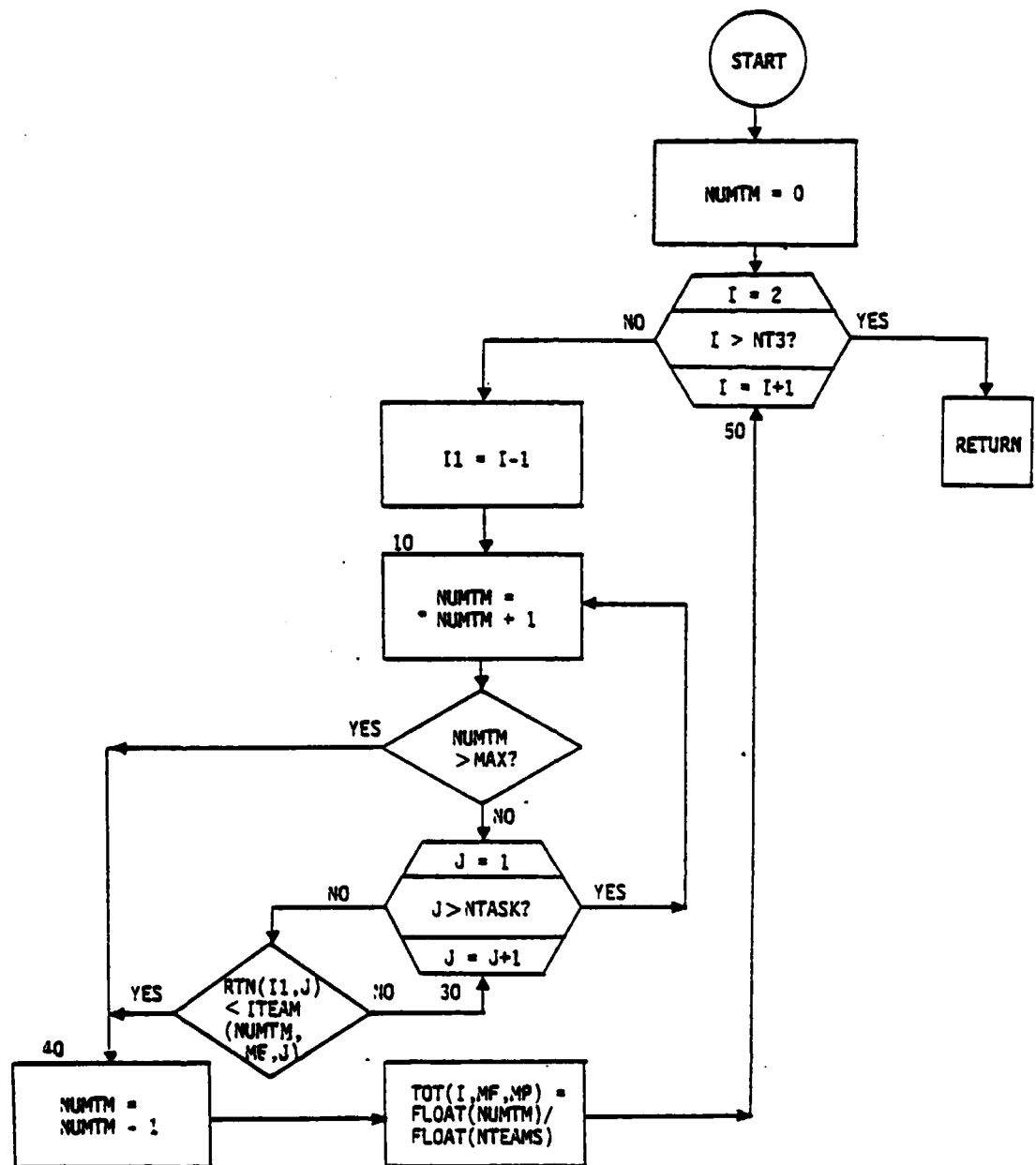


Figure 3-10. Subroutine RCAP

that time. The process is repeated for each time of interest and the resulting capabilities are stored in array TOT by time slice, mission, and personnel or materiel designation.

The argument MAX is the maximum number of teams which can be constructed. ITEAM is a dummy array which contains the personnel or materiel team requirements, as appropriate. RTN is the array by time slice of available assets established by subroutine WHEN. NTASK is the appropriate number of task lines, personnel or materiel. NT3 is the number of times input plus three. TOT is the array of capabilities which are calculated by the subroutine. MF is the mission number and MP designates personnel (1) or materiel (2).

3.10.2 COMMON BLOCKS

GENERL

3.11 SUBROUTINE ASN (MP, MF, NUMTRY)

3.11.1 General

Subroutine ASN (Figure 3-11) files the assignment matrix information for each iteration in DEFINE FILE 22. This subroutine is called by MAIN if the option flag ASSIGN is greater than zero. The calling arguments identify personnel (MP=1) or materiel (MP=2), the mission number (MF), and the number of the team (NUMTRY) which was completed.

Subroutine ASN first converts the integer values of the allocations matrix (IALLO) to real numbers and stores them in array ALLO. The DEFINE FILE record number, KOUNT, and record length, RLGTH, are then calculated. Each record is defined by team number, mission number, and either personnel or materiel. (See paragraph 3.11.3 below for a discussion of the DEFINE FILE structure.) The appropriate record

is then read into array WORK and the data for this iteration is summed with any previously stored data. WORK (1) is then incremented as a count of the number of iterations represented by the data. The array WORK is then written back into the appropriate DEFINE FILE record and control is returned to MAIN.

3.11.2 COMMON BLOCKS

GENERL
WK1
WK2

3.11.3 DEFINE FILE 22

(1) Records are identified by team number, mission number, and type (personnel or materiel). These records are filed in order as follows: personnel team number from 1 to NTEAMS, mission 1, followed by materiel teams from 1 to NTEAMS, mission 1. Each additional mission follows in the same order.

(2) Data in each record is accumulated for all iterations that team is constructed. The data in each record is filed as follows:

- Element 1: The number of iterations represented.
- Element 2: Redundant, equal to #1.
- Elements 3-(NTASK+3): The assignments made of resources in task #1 to the demands of each task or to surplus.
- Elements (NTASK+4)-(2xNTASK+5) and subsequent strings of length NTASK+1 are the assignment data for each task, 2 thru NTASK, to fill any other task or to surplus. For materiel the array is further extended for the assignment data of lightly and then moderately damaged items.

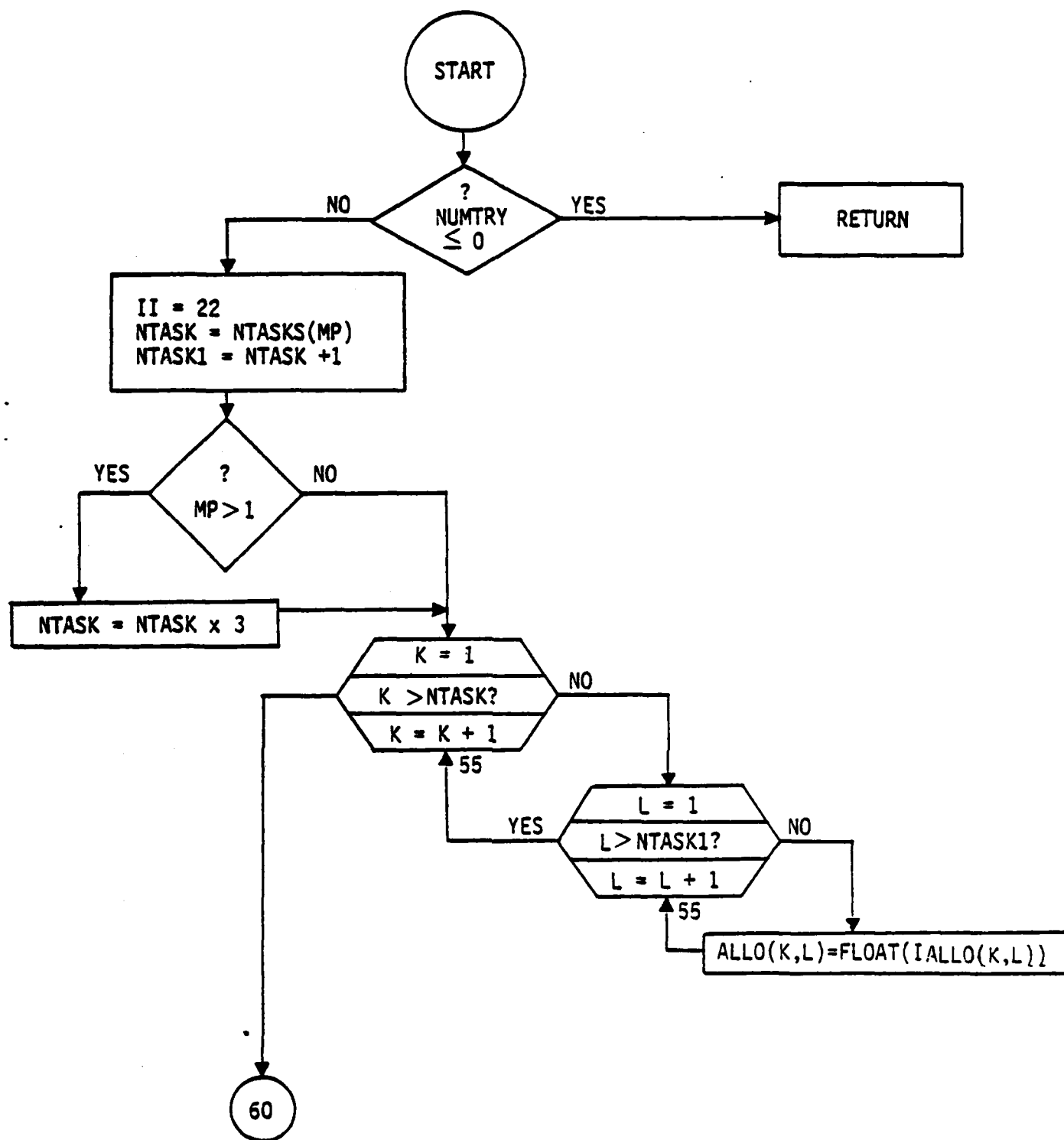


Figure 3-11. SUBROUTINE ASN

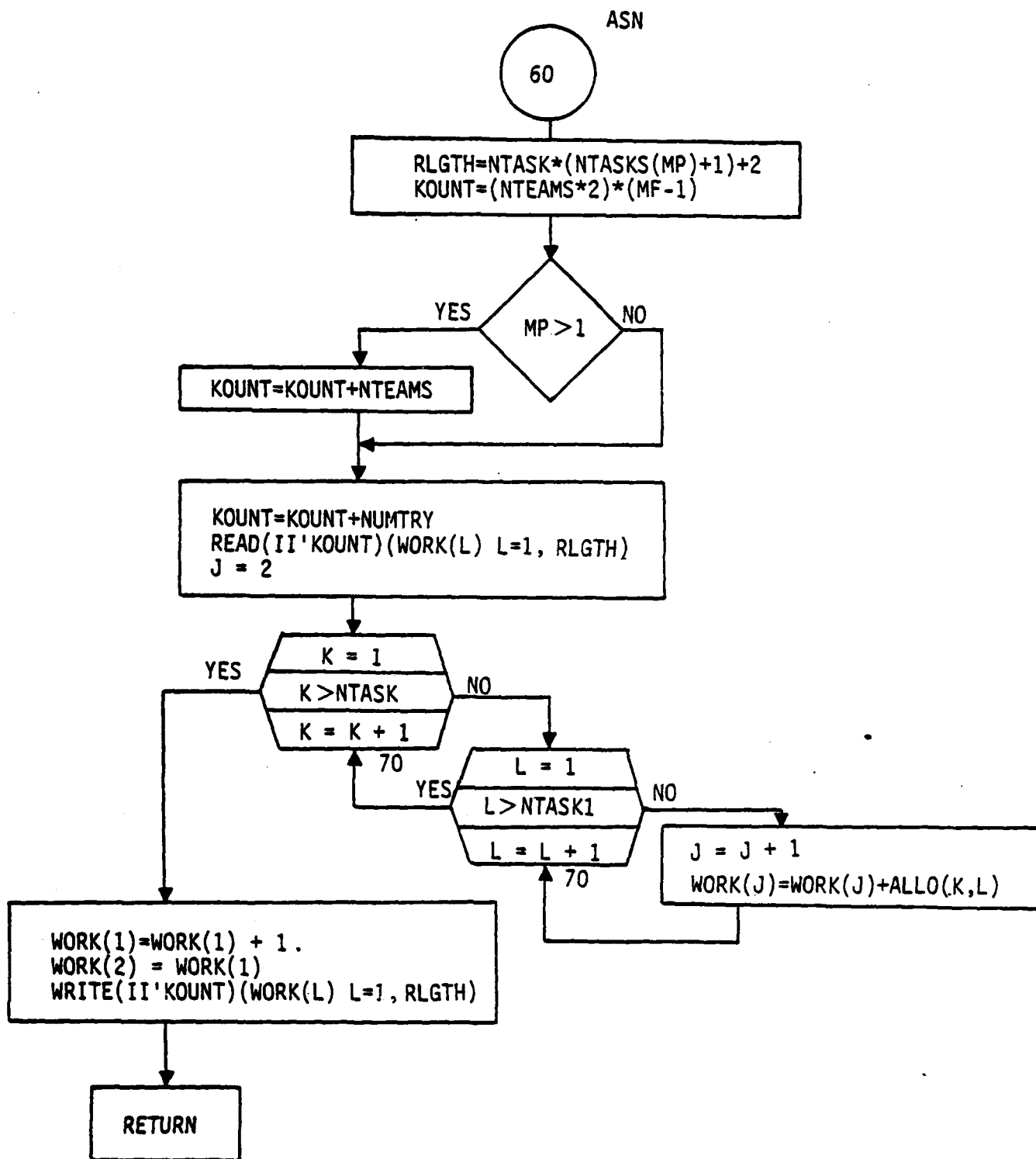


Figure 3-11. Subroutine ASN (Continued)

3.12 SUBROUTINE CHOKE (MP, MF, NUMTRY)

3.12.11 General

Subroutine CHOKE (Figure 3-12) calls subroutine TRANS to solve the transportation problem given the requirements of the next higher team number. This identifies those individuals or items which are most critical to the reconstitution of higher capability. Subroutine CHOKE then stores this data in DEFINE FILE 21. Records are identified by team and mission number for personnel or materiel. Data in each record consists of the number of iterations, the number of alternate solutions found (when that option is exercised), and the needs and surplus data developed from the allocation matrix which has been completed by subroutine TRANS. The detailed structure of the DEFINE FILE is given in paragraph 3.12.3 below.

Subroutine CHOKE is called by MAIN if the input option flag, SCHOKE, is greater than 0. The call is made each damage replication, each mission for both personnel and materiel. The calling arguments identify personnel or materiel (MP), the mission number (MF), and the number of the largest team which could be constructed from the available assets (NUMTRY).

Within the subroutine, the variable IFLG is used to indicate if all teams attempted were constructed (IFLG=1). In that case there will be no shortage and only data of surplus items is calculated. The variable MULTF is an input option flag and if greater than zero a call will be made to ENTRY ALTOPT, in subroutine TRANS, to attempt to find alternate optimal solutions.

When subroutine TRANS solves the transportation problem, in this case known to be infeasible, it will use either dummy supply or infeasible assignments with a very large cost. In either case the

assignment is easily identified and the demand filled by that assignment is critical to, "chokes", the construction of that team.

The variable NOTEN, calculated in TRANS, is the number of dummy supply items required to complete the transportation problem. If NOTEN is zero then no dummy supply was used and all "choke" assignments were made using assets on hand. These assignments are identified by a cost of MBIG, a large value which is calculated by subroutine COSTMM. Any assignment which has been made in this way identifies a need and is stored in array AVEN by task number. Since this assignment is, in fact, not real the item that was assigned is actually surplus and is therefore also added into the surplus array, AVES, by task number. In the case of materiel this array is extended for each task's light and moderate maintenance categories.

In the case where NOTEN has some value all needs are filled by the assignment of dummy supply. It is therefore only necessary to examine the dummy supply row (NTASK+1 or for materiel 3xNTASK+1) for "choke" assignments.

Both surplus (AVES) and need (AVEN) arrays are constructed. The WORK array is read from the appropriate record in the DEFINE FILE and data from this iteration is accumulated into array WORK. This array is then written to the appropriate DEFINE FILE record for later use. (See paragraph 3.12.3 for DEFINE FILE structure.)

When alternate optimal solutions are found the average value for both surplus and "choke" assignments are calculated on each iteration. Additionally, the minimum and maximum value of each for all the solutions found for a particular team are calculated.

3.12.2 COMMON BLOCKS

DLY3
GENERL
KTR1
KTR2
SURV
WK1
WK2

3.12.3 DEFINE FILE 21

(1) Records are identified by team number, mission number, and by type (personnel or materiel). The records are filed in the following order: personnel teams, from 1 to NTEAMS+1, followed by materiel teams, from 1 to NTEAMS+1. This sequence is repeated for each mission. The additional (+1) team count provides space for the case when all teams are constructed and the "choke" data represents the surpluses after the last team is built.

(2) Each record is an accumulation of data for all iterations which "choke" on that team number. Data is stored in each record as follows:

- Element 1: The number of iterations represented.
- Element 2: The total number of solutions found. If multiple optimal solutions are not examined or not found this value will be equal to Element 1.
- (a) No alternate optimal solutions considered:
(Note: NTASK is the number of personnel tasks or 3 times the number of materiel tasks as appropriate)
 - Elements 3 -(NTASK+2): Needs for each task
 - Elements (NTASK+3)-(2NTASK+2): Square of the needs for each task.
 - Elements (2NTASK+3)-(3NTASK+2): Surpluses for each task.
 - Elements (3xNTASK+3)-(4xNTASK+2): Square of the surplus for each task.
 - The record for maximum teams +1 has data only for surpluses. In that case the surplus data

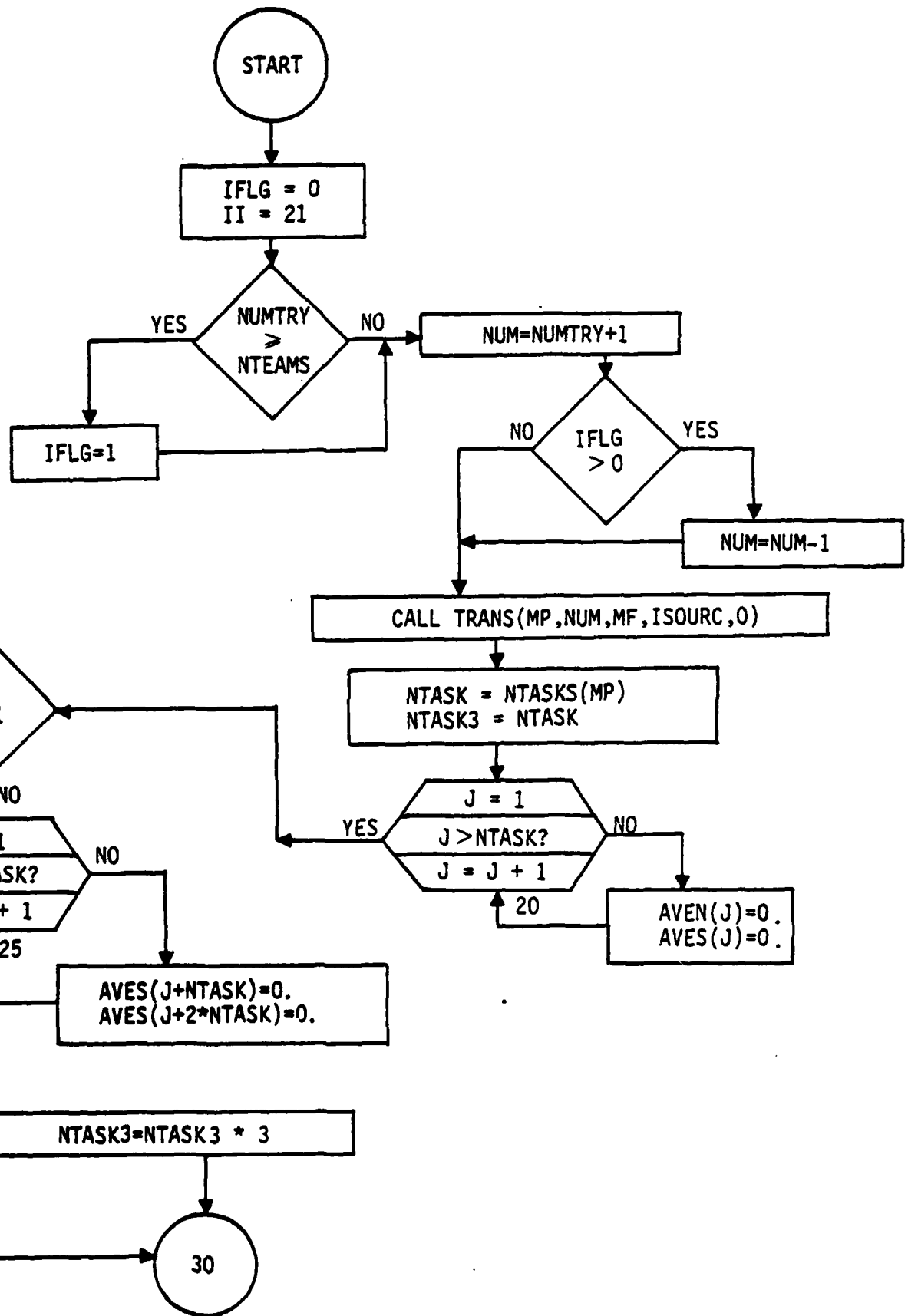


Figure 3-12. Subroutine CHOKe

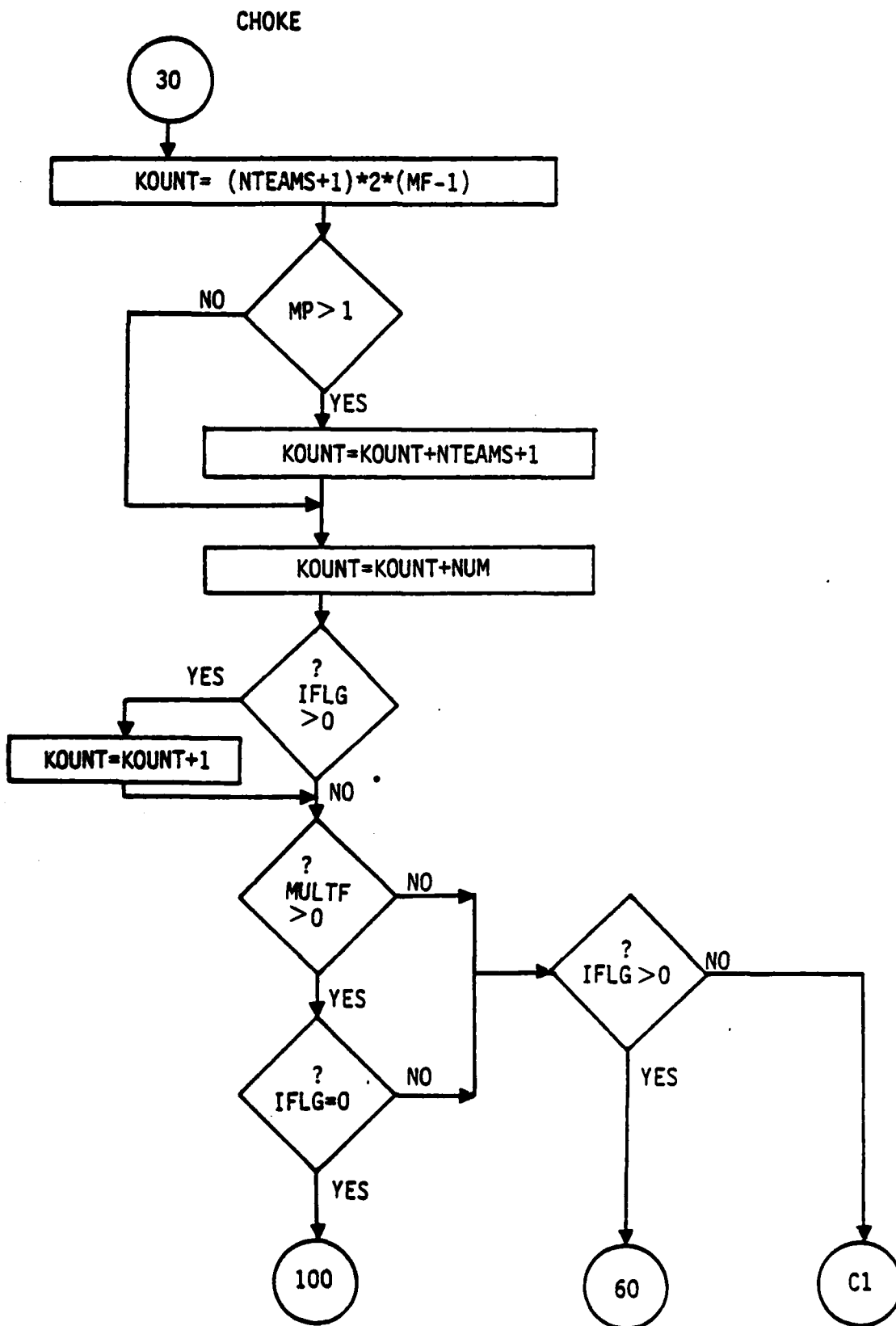
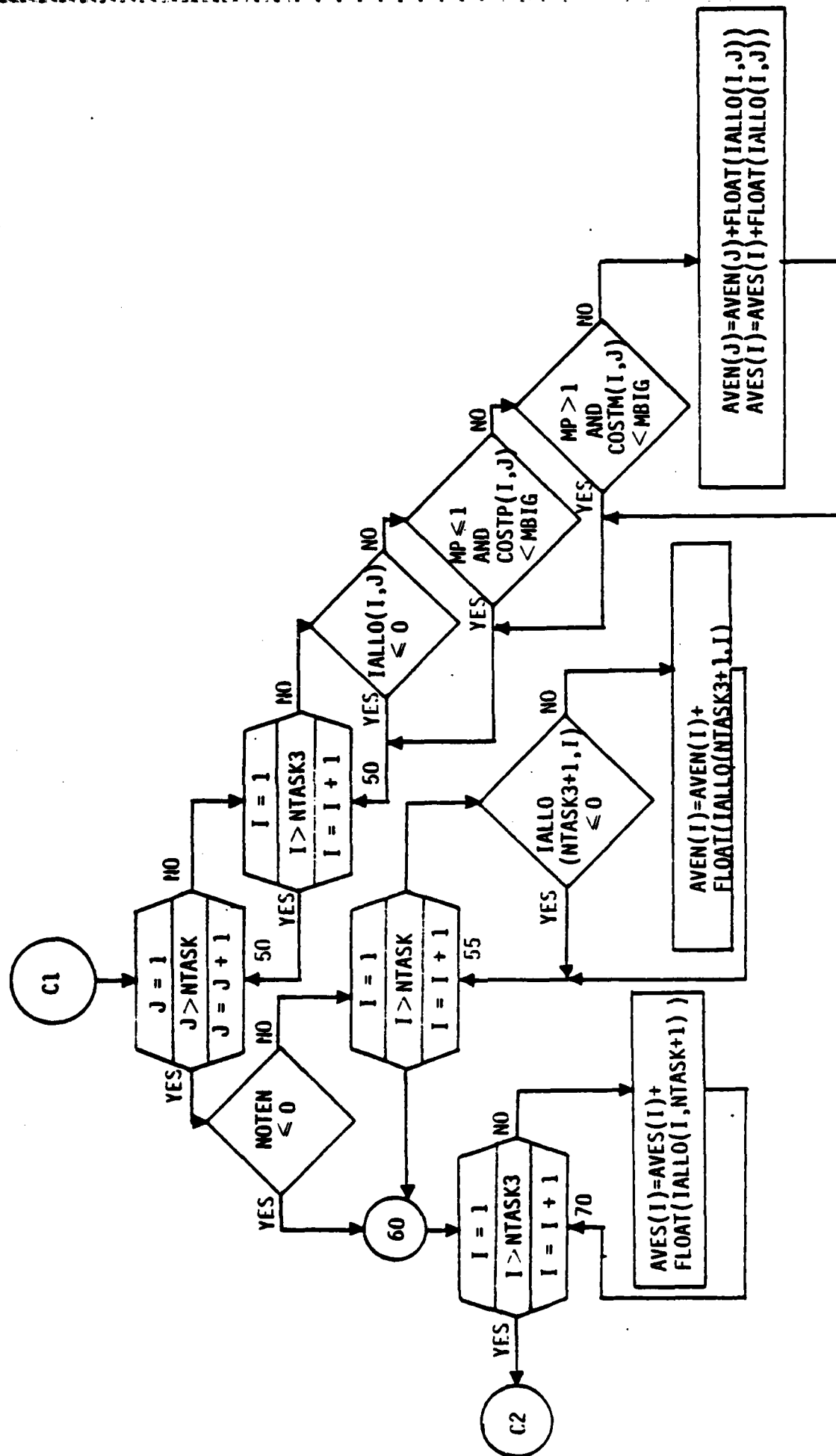


Figure 3-12. Subroutine CHOKe (Continued)



3

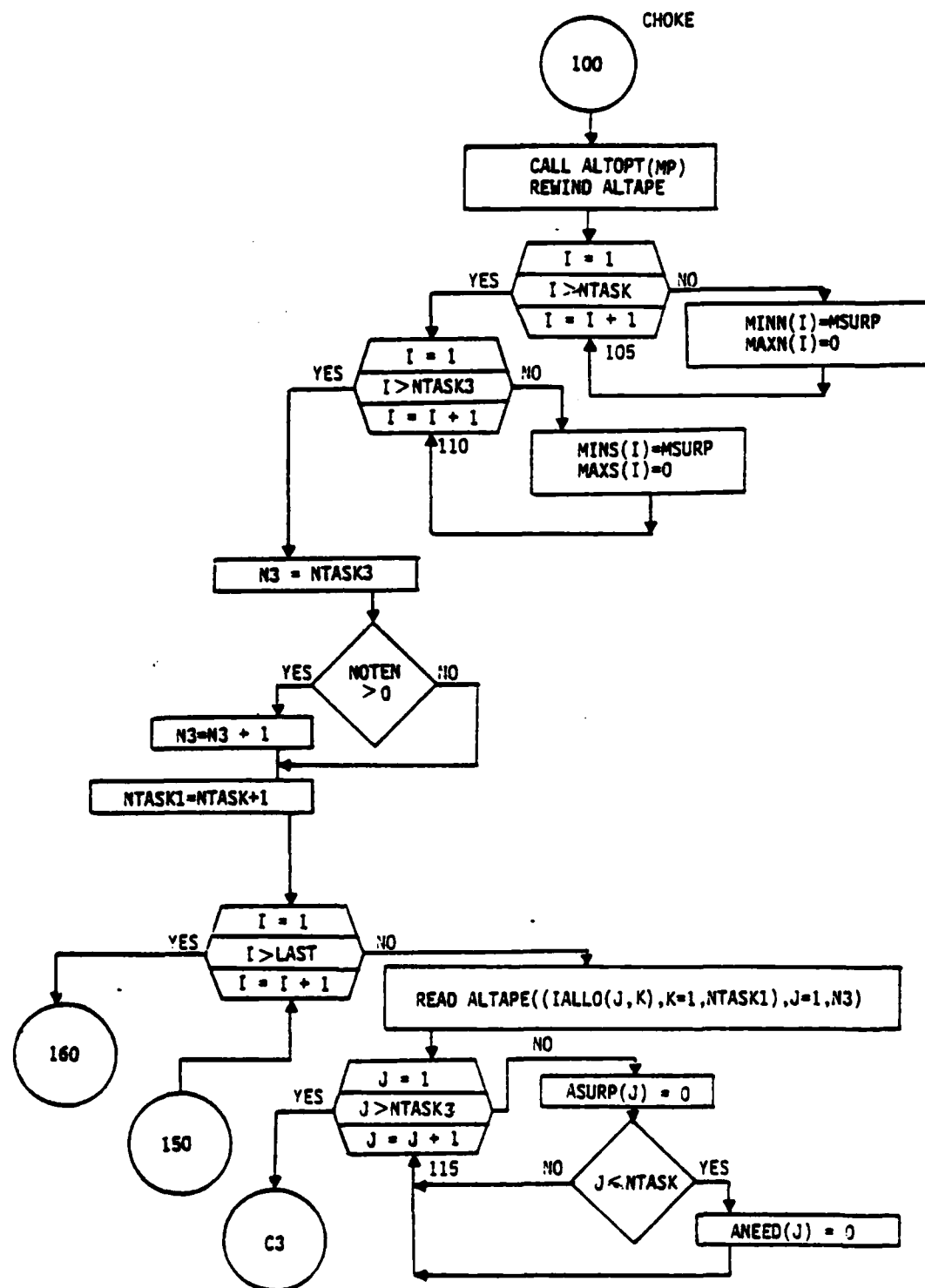


Figure 3-12. Subroutine CHOK (Continued)

160

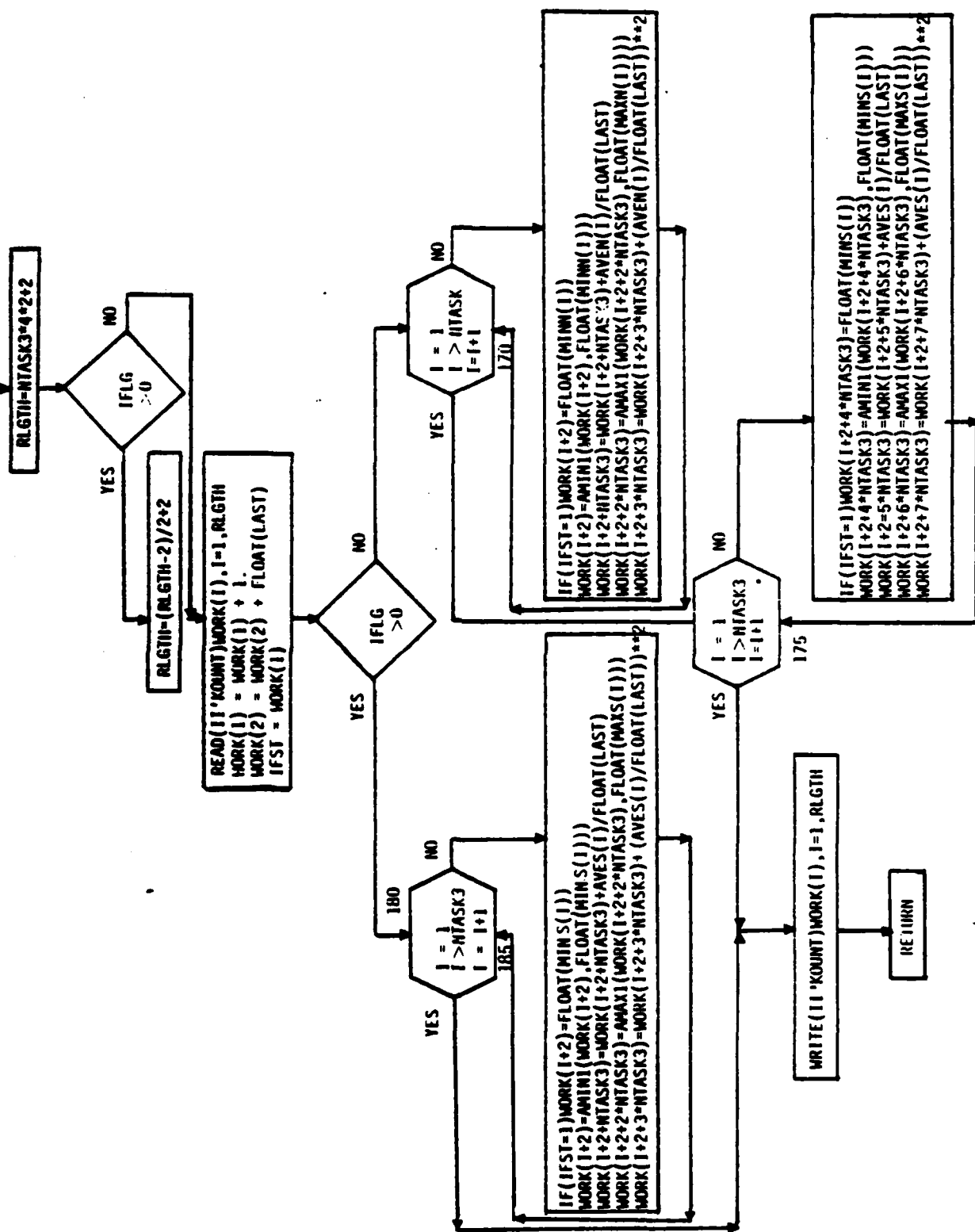


Figure 3-12. Subroutine CMLKE (Continued)

CHOKE

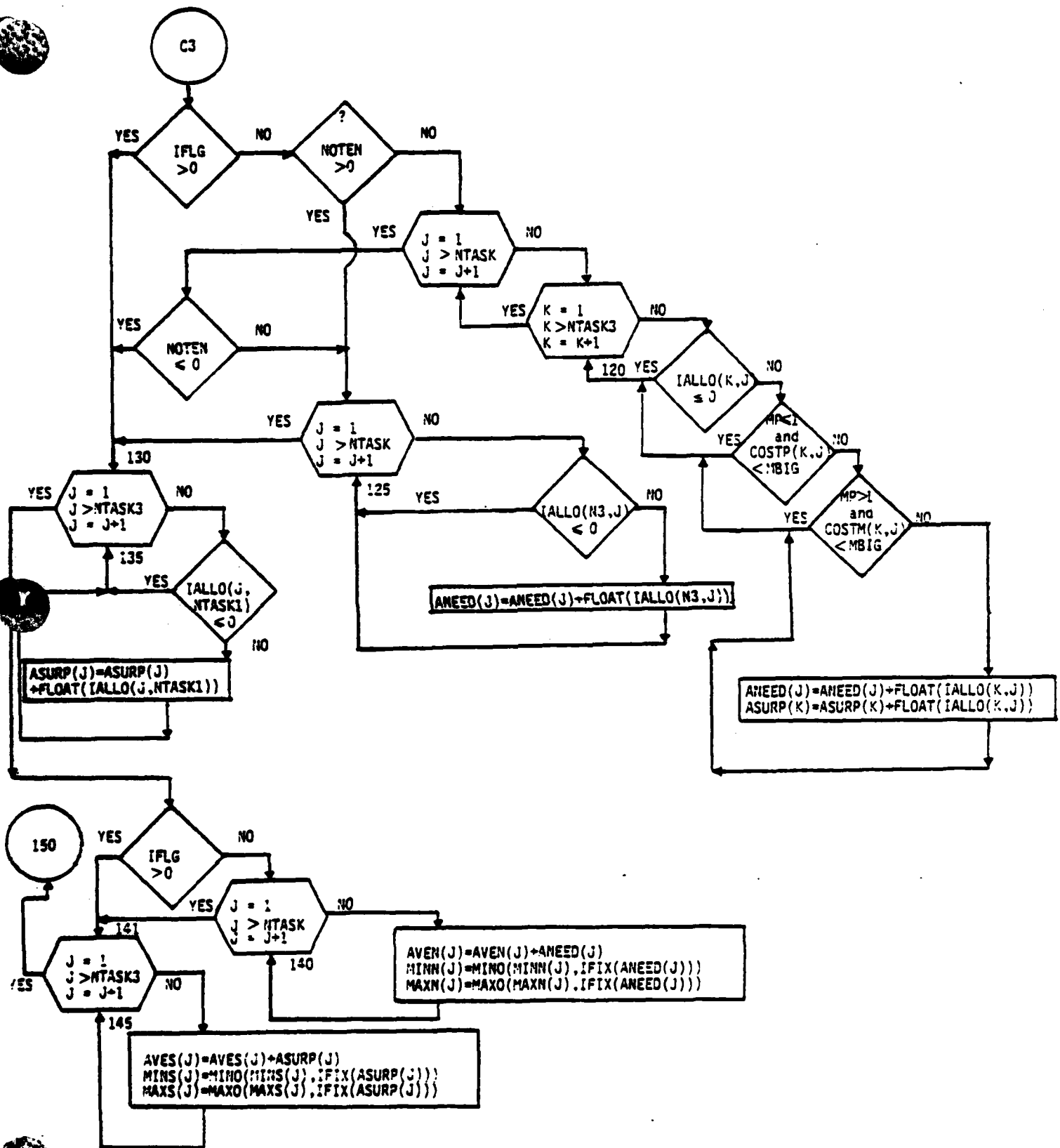


Figure 3-12. Subroutine CHOKE (Continued)

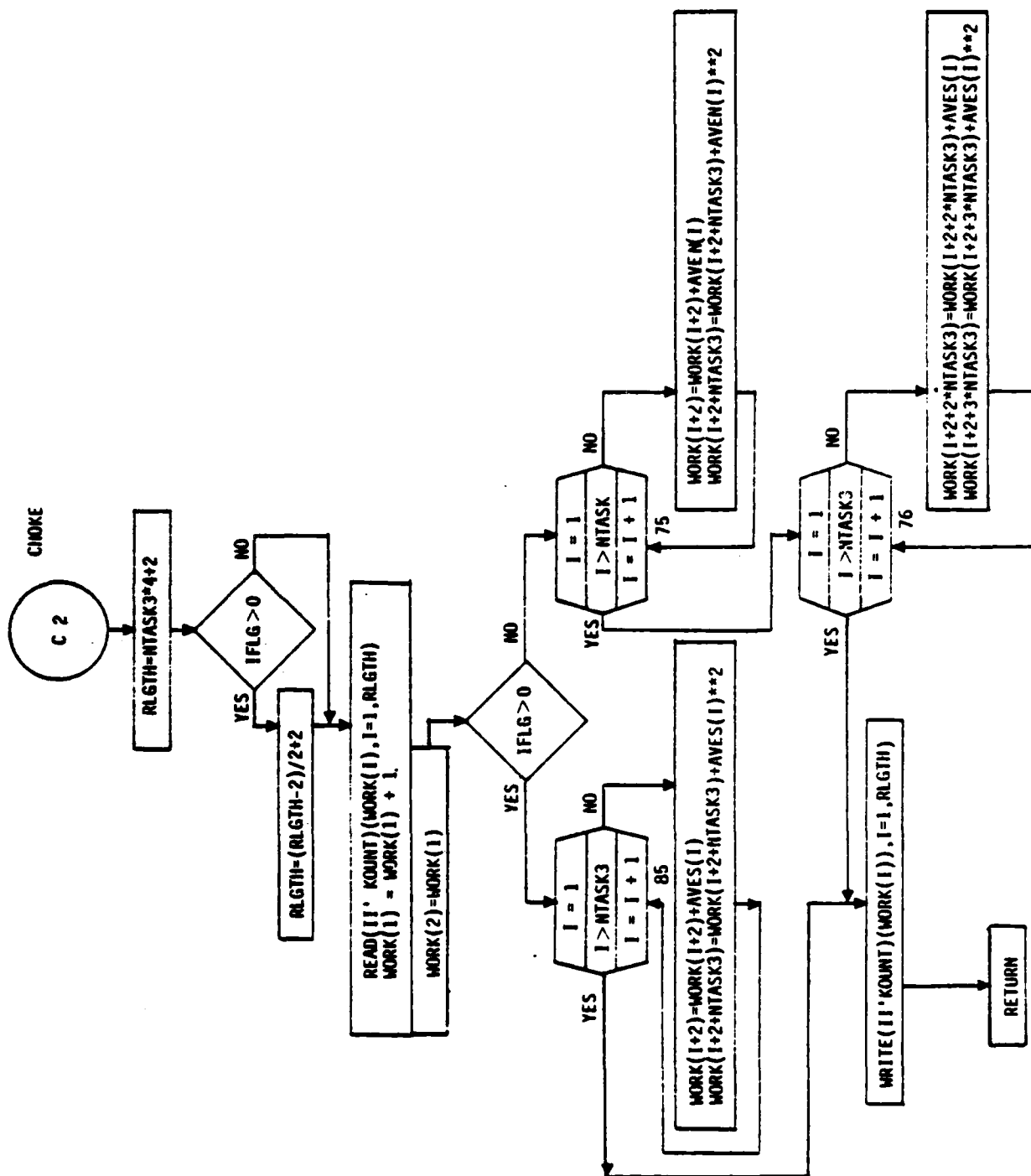


Figure 3-12. Subroutine CJOKE (Continued)

is written in the first two element strings, the third and fourth strings are not used.

(b) Alternate optimal solutions are considered:

- Elements 3-(NTASK+2): For each task, the minimum value need from all solutions found. (not accumulated)
- Elements (NTASK+3)-(2xNTASK+2): Average need, each task, for the solutions found.
- Elements (2xNTASK+3)-(3xNTASK+2): The maximum value need, each task, from all solutions found (Not accumulated)
- Elements (3xNTASK+3)-(4xNTASK+2): Square of the average need, each task, for the solutions found.
- Elements (4xNTASK+3)-(5xNTASK+2): The minimum value surplus, each task, for all solutions found (Not accumulated).
- Elements (5xNTASK+3)-(6xNTASK+2): The average surplus, each task, for the solutions found.
- Elements (6xNTASK+3)-(7xNTASK+2): The maximum value surplus, each task, for all solutions (Not accumulated).
- Elements (7xNTASK+3)-(8xNTASK+2): The square of the average surplus, each task, for all solutions found.
- The record of maximum teams +1 has data only for assignment to surplus. In that case surplus data is written into the first four element strings and strings five thru eight are not used.

3.13 SUBROUTINE STAT (TMEAN, TOTCAP, SD, GMEAN, GSD)

3.13.1 General

Subroutine (Figure 3-13) is the last subroutine called in the damage iteration loop and is called immediately following completion of the mission loop and personnel/materiel loop. The primary function of subroutine STAT is to accumulate capability data for all iterations. The data is then further processed by subroutine OUTD when all iterations have been completed.

STAT (TMEAN,TOTCAP,SD,GMEAN,GSD)

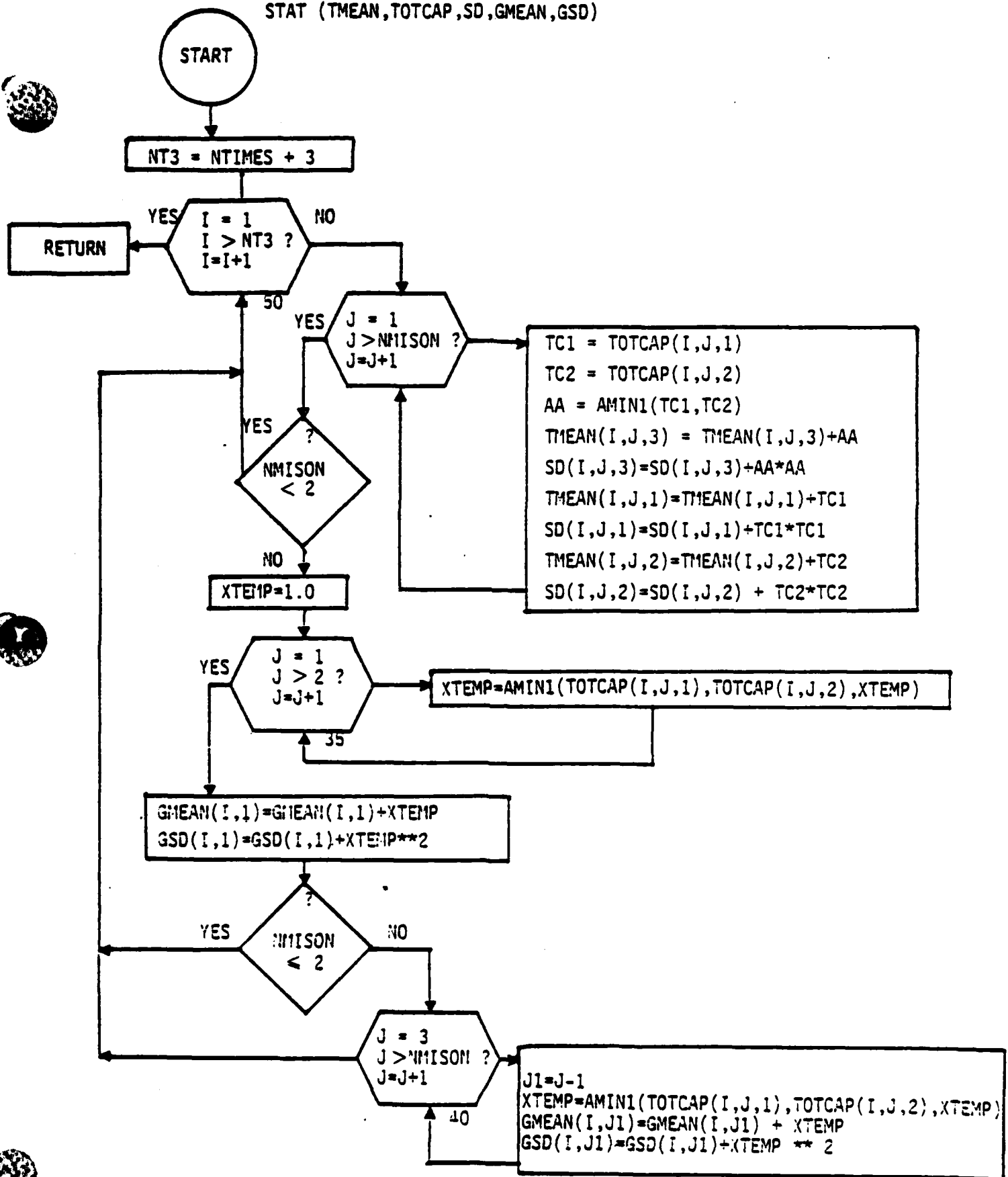


Figure 3-13. Subroutine STAT

Capability data from array TOTCAP is accumulated into array TMEAN for both personnel and materiel. The minimum value of the personnel or materiel capability is also accumulated into array TMEAN and represents the maximum unit capability. The squares of each of the capability values are summed into array SD for use in calculating the confidence interval. All of the above arrays are indexed by time period number, mission number, and personnel (1), materiel (2) or unit (3) indicator.

Capability is also calculated for combinations of missions, if there is more than one. This value is the minimum capability of the missions considered. Array GMEAN is used to accumulate these values. The square of each calculated capability is accumulated into array GSD. Both of these arrays are indexed by time period and mission combination indicator: 1=Mission 1 and 2; 2=Mission 1, 2, & 3; etc.

3.13.2 COMMON BLOCKS

GENEAL

3.14 SUBROUTINE OUTD (TMEAN, SD, GMEAN, GSD)

3.14.1 General

Subroutine OUTD (Figure 3-14) is called by the MAIN routine after all replications of a particular damage set have been completed. This routine calculates the average capability and confidence interval at each time of interest. Calculations are made for personnel, materiel, unit, and mission combinations of unit capability. The routine also calculates the integral (using the trapezoidal rule) of the unit and mission combination capabilities as a function of time. Calculations are made using data accumulated by subroutine STAT on each iteration. The arrays TMEAN and GMEAN contain summations of capabilities computed each iteration. These data are simply divided

by the number of iterations to get the average values. Averages are calculated in array TEMP for printing. The arrays SD and GSD contain summations of the square of the capability each iteration and are used for calculation of the confidence interval. These values are calculated in array STEM P for printing.

The confidence intervals are based on the t test of significance for a 90 percent confidence level. The basic equation is:

$$90 \text{ percent Confidence Interval} = t_{(n)} \sqrt{\frac{\sum X_i^2 - (\sum X_i)^2 / N}{N(N-1)}}$$

where: X_i = capability for iteration i

N = number of iterations

$t_{(n)}$ = table value of t for N-1 degrees of freedom

Values of $t_{(n)}$ are input by a data statement in the subroutine. The table of these values is not extensive but is sufficient to provide reasonable accuracy in the calculations.

Calculations are performed and printed for one or two missions at a time, determined by how many missions are included. A maximum of two missions can be printed per page and mission 1 does not require the calculation or print of combined mission capability. Thus four separate calculation and print loops are required.

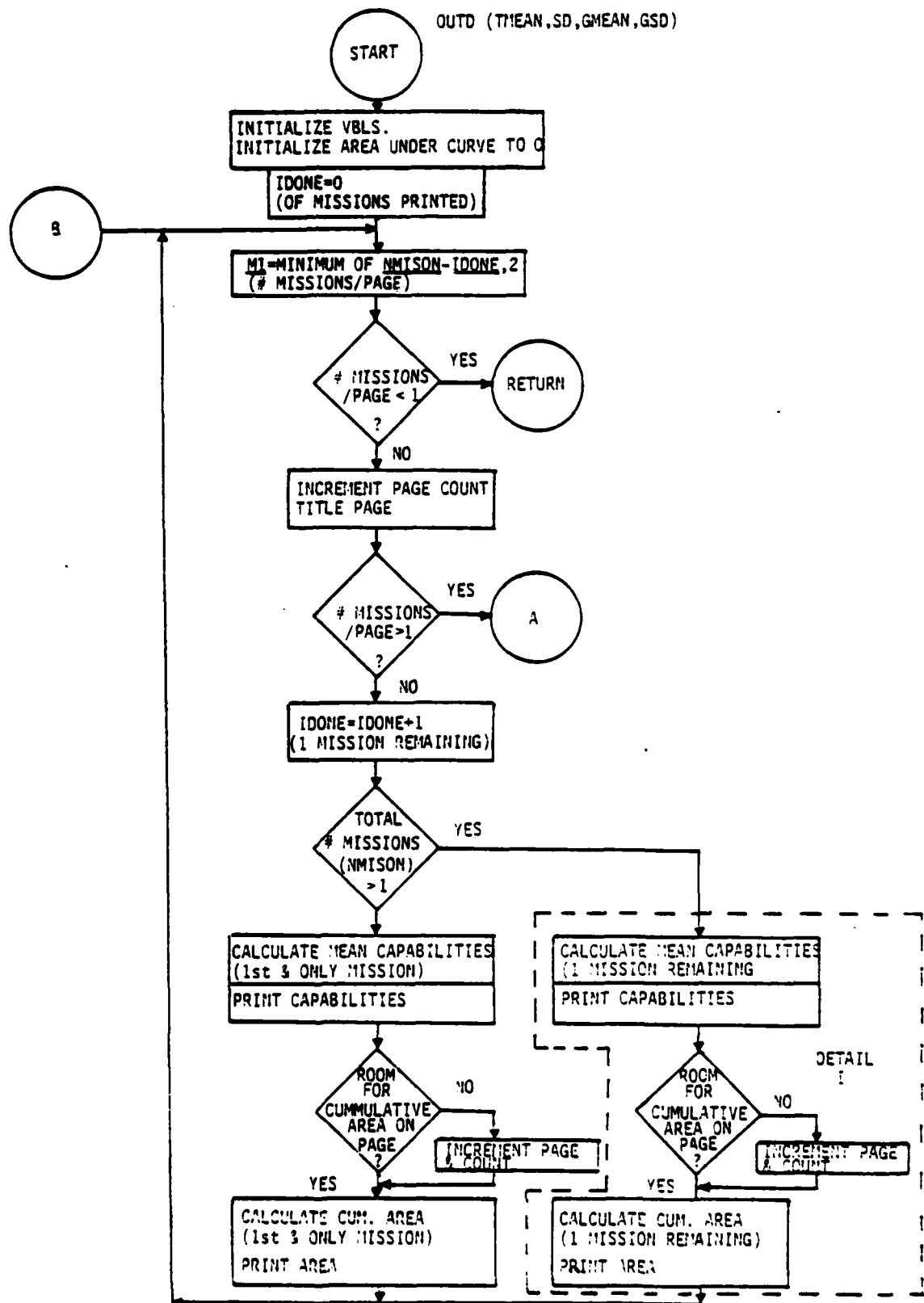


Figure 3-14. Subroutine OUTD

OUTD

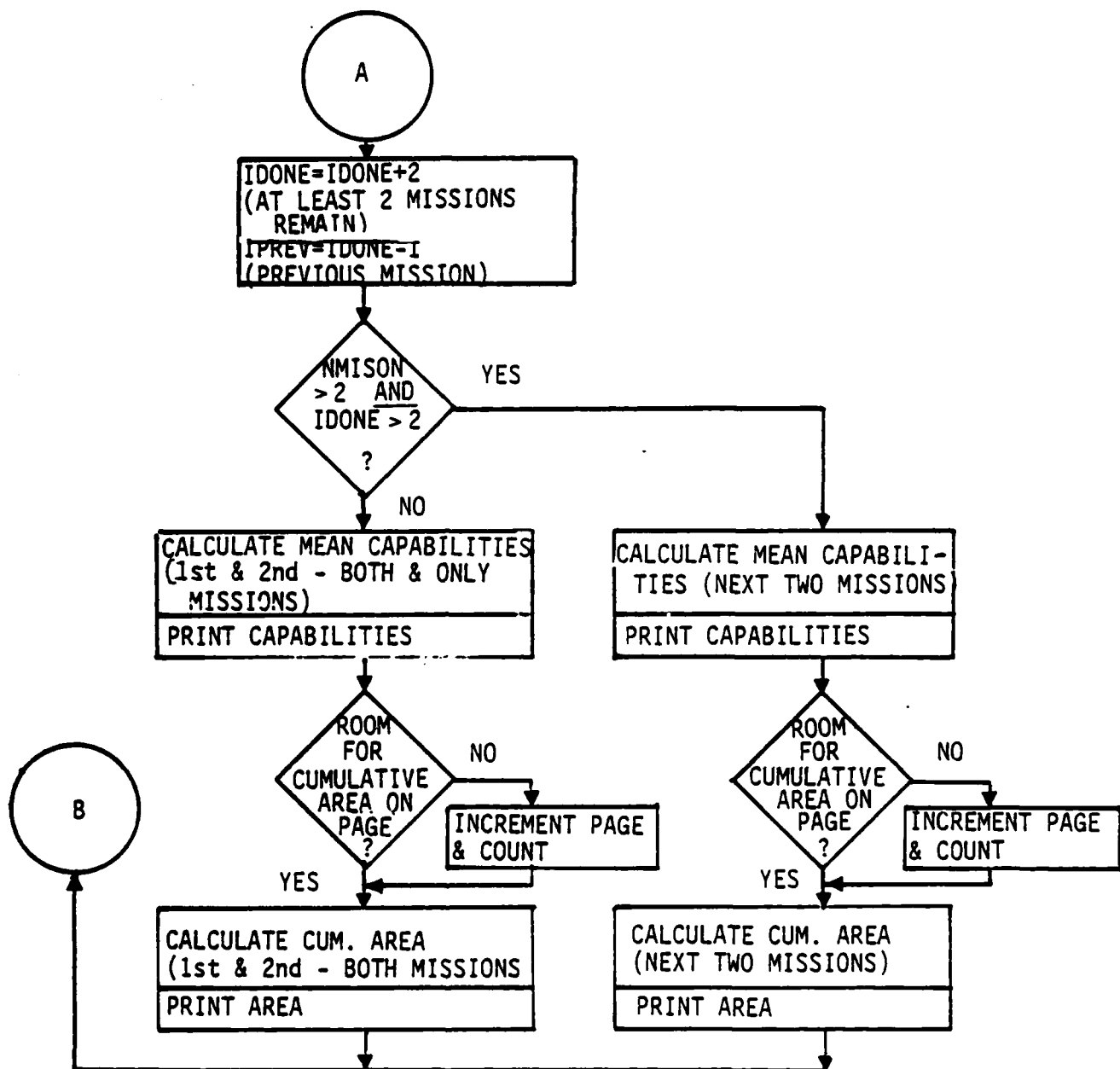


Figure 3-14. Subroutine OUTD (Continued)

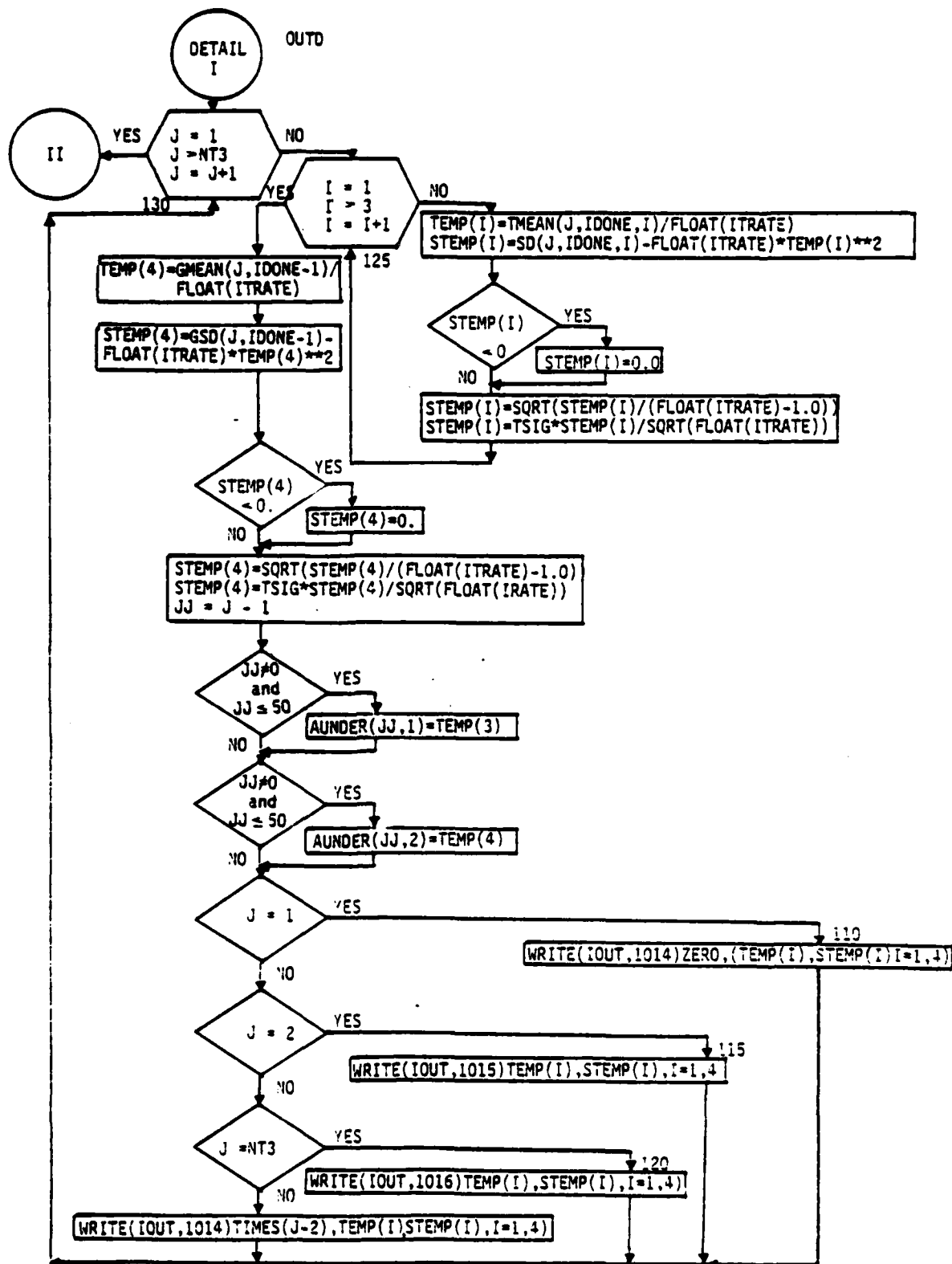


Figure 3-14. Subroutine OUTD (Continued)

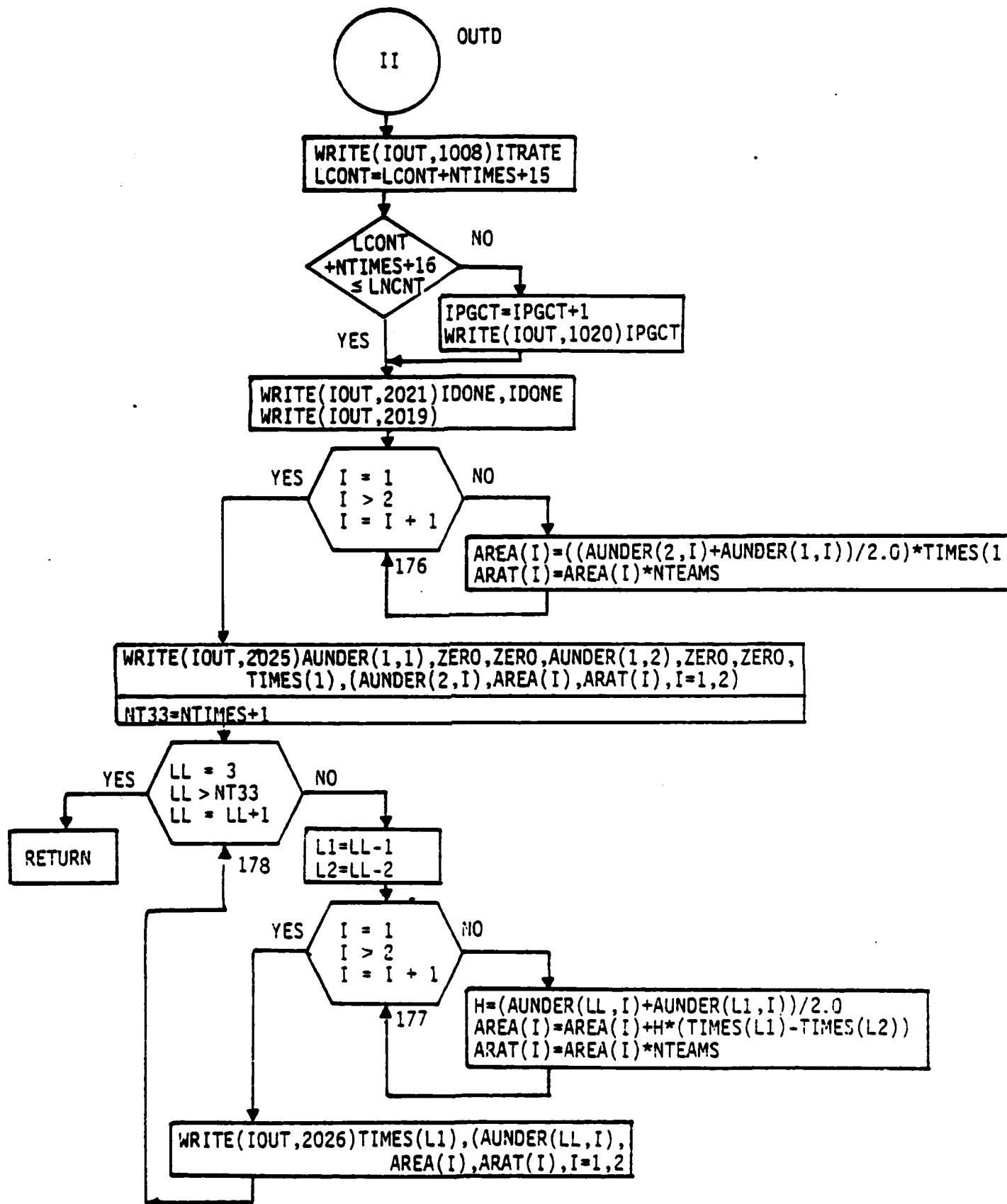


Figure 3-14. Subroutine OUTD (Continued)

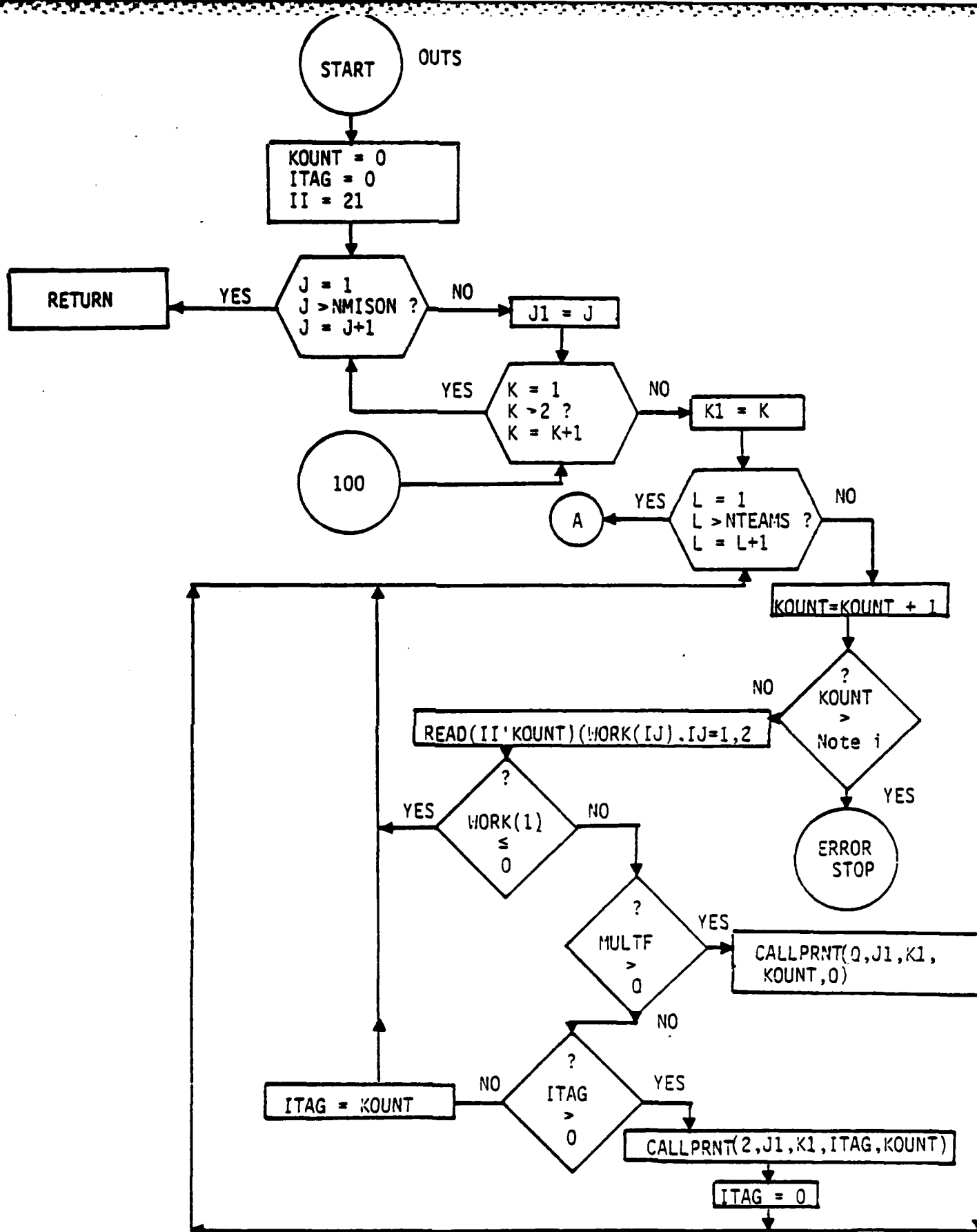
3.15 SUBROUTINE OUTS

3.15.1 General

Subroutine OUTS is called by MAIN if the option SCHOKE is greater than zero. Subroutine OUTS controls the printing of the "choke" data output (Sensitivity Analysis Needs and Surplus). This is accomplished by reading the first two elements (WORK(1) & WORK(2)) of each record in DEFINE FILE 21 to see if data exists in that record. The subroutine operates through a set of nested loops over missions, personnel/materiel, and team number, with the addition of one after the team loop, for those occasions when all teams were constructed.

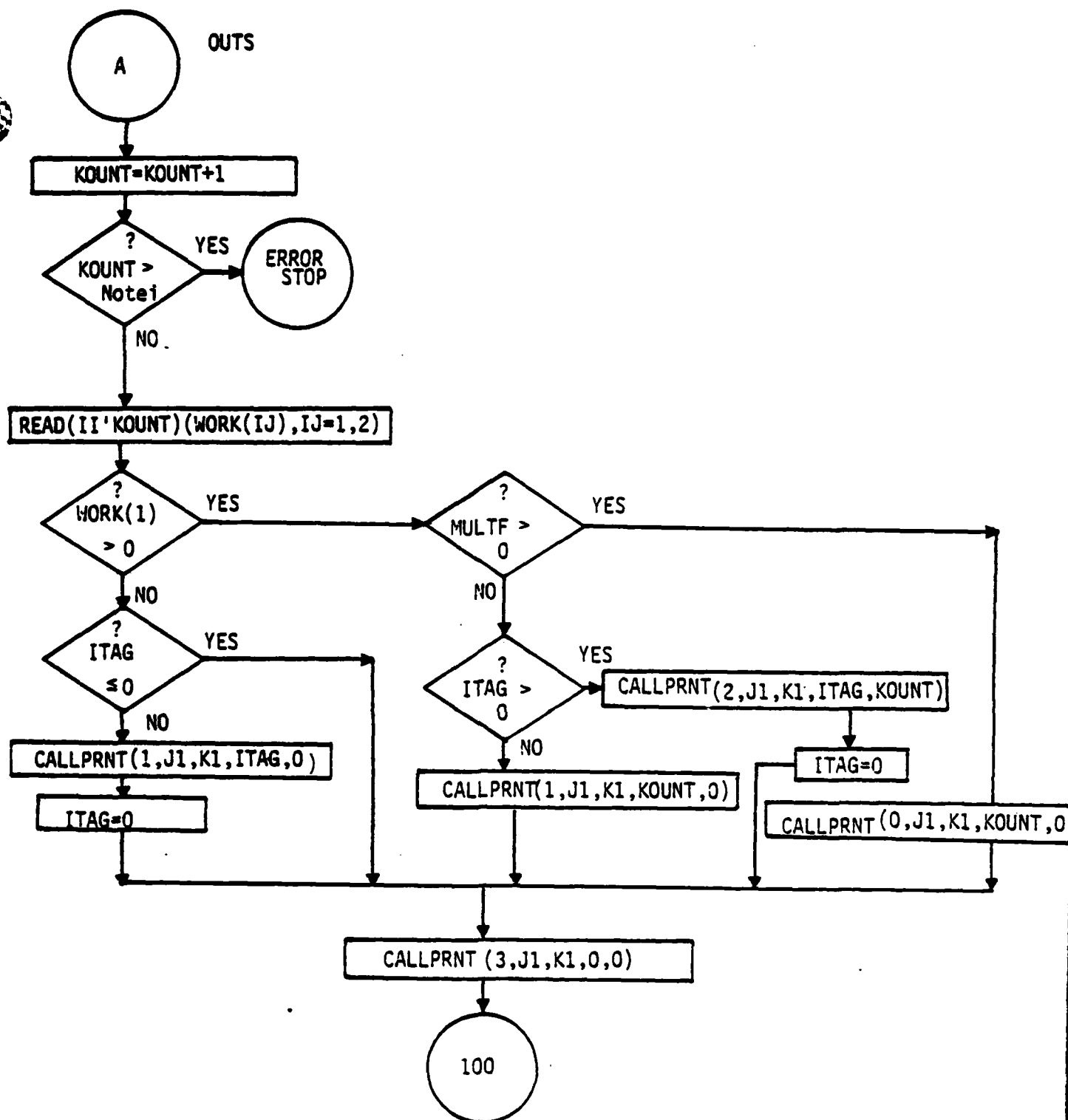
The variable WORK(1) is a count of the number of iterations for which data is stored in that particular record. If alternate optimal solutions were desired, MULTF greater than zero, WORK(2) is a count of the total number of solutions represented by the data. If WORK(1) is zero then the next record is read until a record with data is found. The regular output format will accommodate output for two teams per page. When alternate optimal solutions are desired, an expanded output format is used and only one team per output page can be printed. Therefore, if MULTF>0 subroutine PRNT is called each time data is located. If MULTF=0 the record number, KOUNT, is stored in ITAG until a second record with data is located or until all team numbers plus one have been searched.

The call to subroutine PRNT is made with five arguments. The first argument has no variable name in OUTS but is received by PRNT as IFLG. Its values and their meanings are: 0-multiple optimal solution format is required; 1-output for only one team, regular format, is required; 2-output for two teams, regular format, is required; and 3-end of team output format required.



Note i: $(NTEAMS-1) \times 2 \times NMISON$

Figure 3-15. Subroutine OUTS



Note i = (NTEAMS+1)X2XNMISON

Figure 3-15. Subroutine OUTS (Continued)

The second and third arguments, J1 and K1, indicate the mission number and personnel (1) or materiel (2). The fourth and fifth arguments pass the record number of data to be used by subroutine PRNT. Both of these arguments are therefore used only when the first argument is 2. When the first argument is 0 or 1 then ITAG or KOUNT, as appropriate, is passed as the fourth argument and the fifth argument is zero. Both are zeroed if the first argument is 3.

3.15.2 COMMON BLOCKS

GENERL
KTR2
PRNTIT
WK2

3.16 SUBROUTINE PRNT

3.16.1 General

Subroutine PRNT calculates and prints the "choke" output using data which has been accumulated and stored in DEFINE FILE 21 by subroutine CHOKE (para. 3.12). Subroutine PRNT is called by subroutine OUTS with calling arguments which indicate the type of data to be read and output formatting required (IFLG), the mission number (J), personnel or materiel (K), and record numbers of data to be used (ITEM 1 and ITEM 2). A local variable, ILAST, is used in conjunction with IFLG for further definition of output format requirements. ILAST is given values of 1 to 7 and is used as a pointer for the format of printing top and bottom lines on the output.

Calculations in PRNT are made primarily by the functions SIGMA, WORKFX, SQRTSG, and a duplicate set SIGMA1, WORKF1, and SQRTS1. The data which is read from DEFINE FILE 21 is stored in array WORK for use with the first three functions. If calculations and output of two teams is required, the second data set is stored in array WORK1 for

use with the second set of functions. Discussion will be limited to one case using the array WORK. The precise structure of DEFINE FILE 21 is discussed in paragraph 3.12.3. In general, each record contains data of needs and surpluses, and the squares of each, accumulated for all the iterations that a particular team "choked." When the alternate solution option is exercised the minimum and maximum value for both needs and surpluses is also contained in the record.

Using the record number furnished by OUTS, subroutine PRNT reads the appropriate data record. The function WORFX is used with appropriate data elements to calculate the average need and surplus per iteration for each personnel or materiel line item. The functions SIGMA and SQRTSG are used in the calculation of the standard deviation using the dummy variables SIG1 and SIG2 for intermediate calculations in the process. The calculation of standard deviation uses the general equation:

$$SD = \sqrt{\frac{\sum Xi^2 - N\bar{X}^2}{N-1}}$$

where: N = number of iterations

Xi = value of needs or surplus for iteration i

\bar{X} = average value of needs or surpluses

3.16.2 COMMON BLOCKS

DLY1
 GENERL
 KTR2
 PRNTIT
 WK2

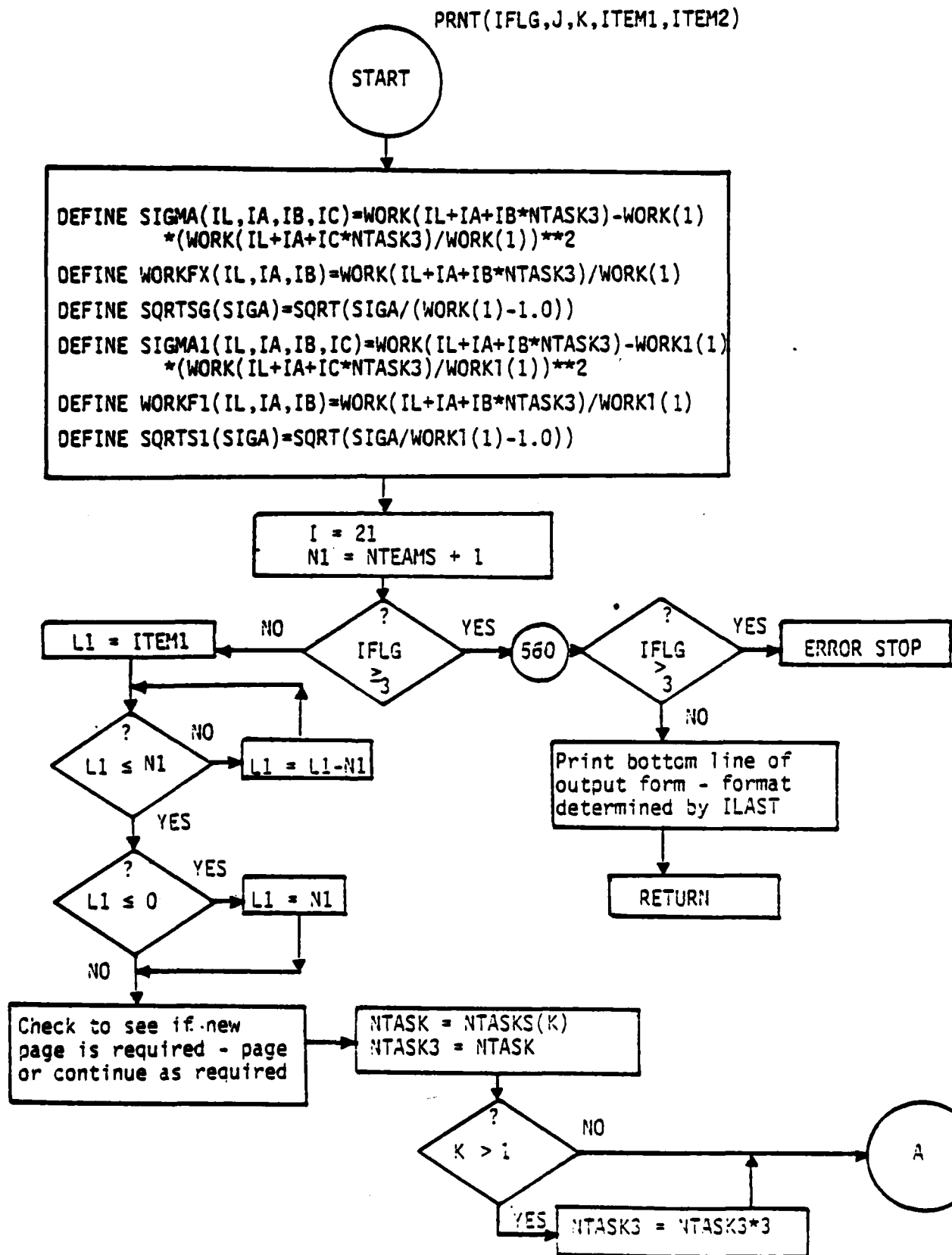
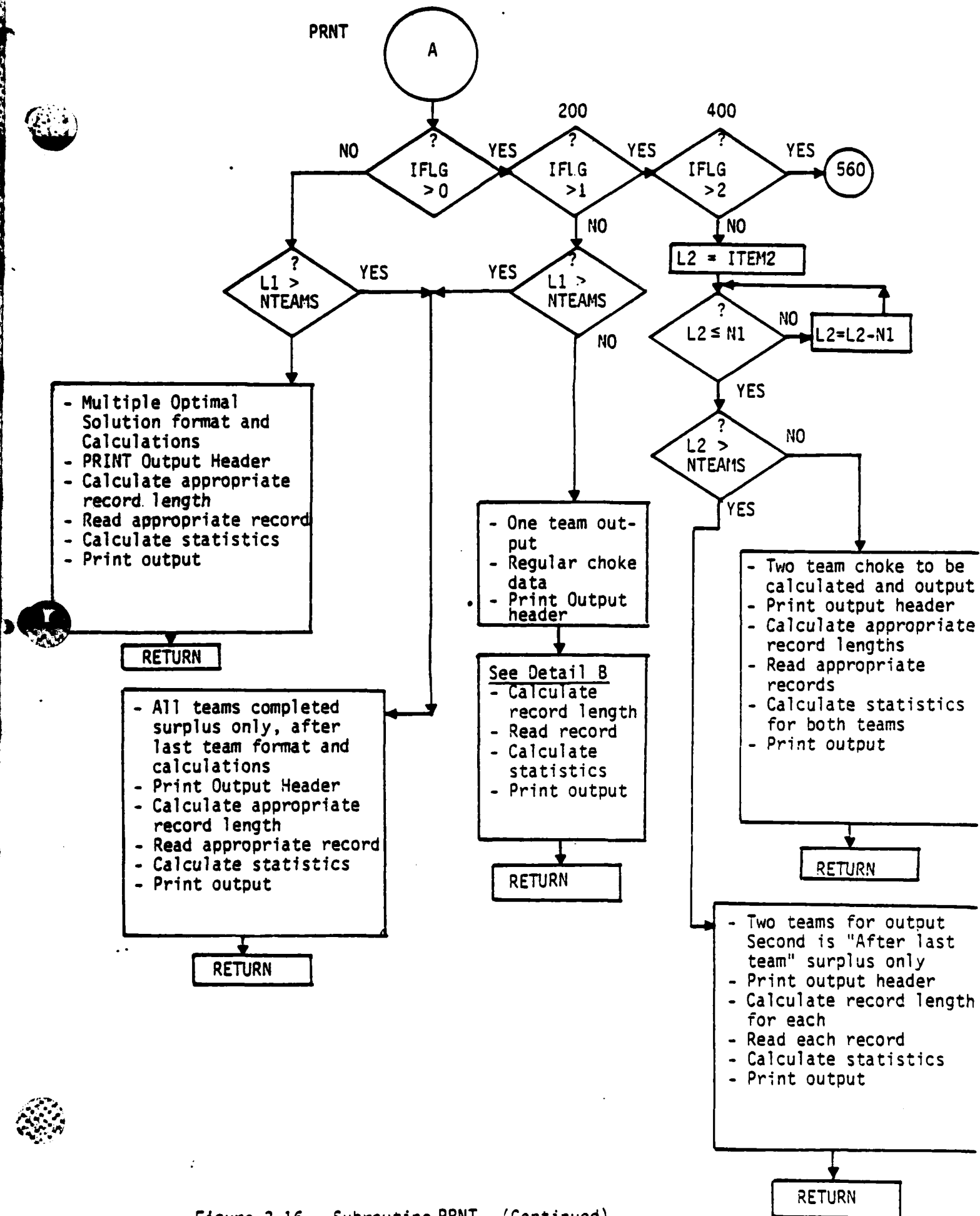


Figure 3-16. Subroutine PRNT



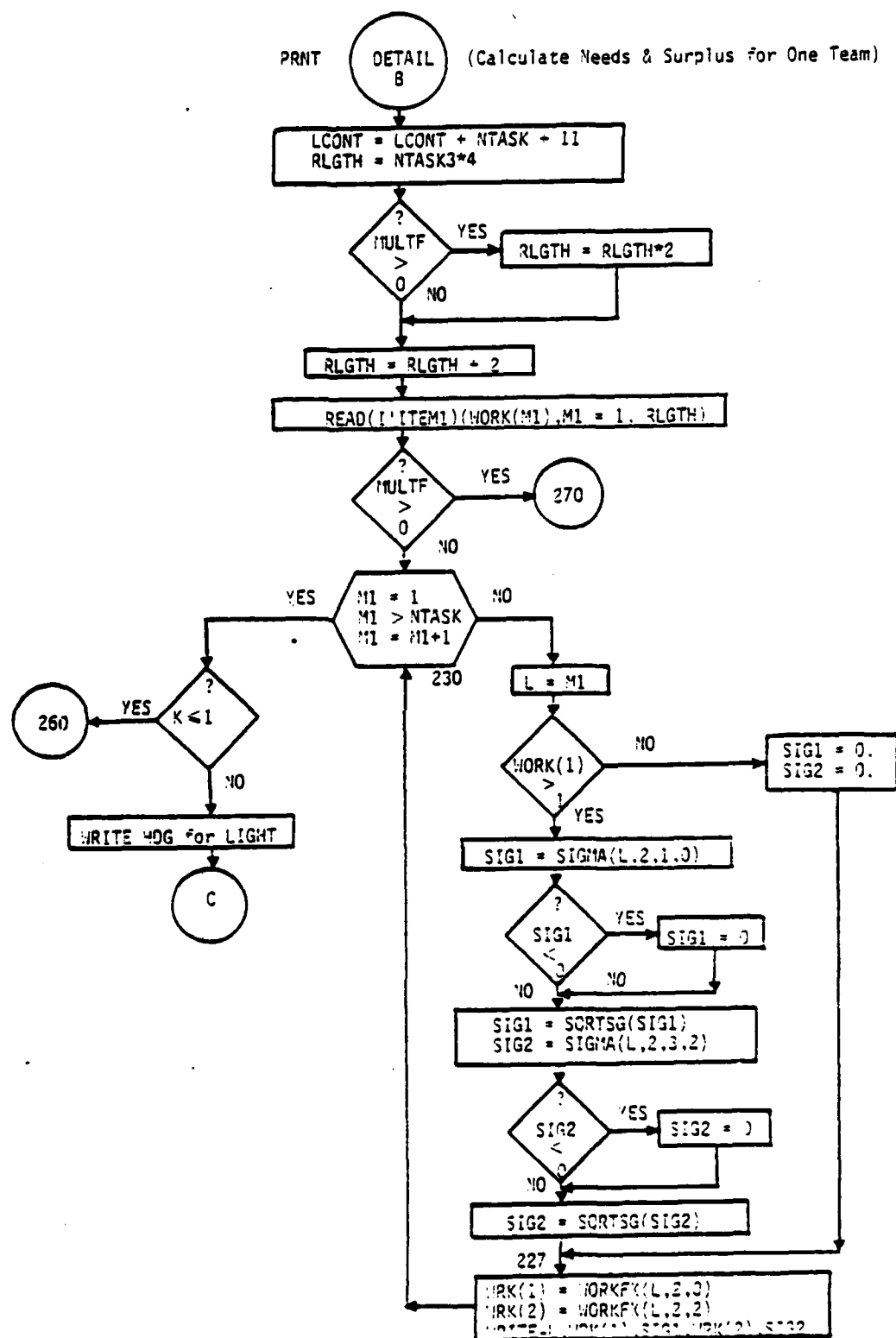


Figure 3-16. Subroutine PRNT (Continued)

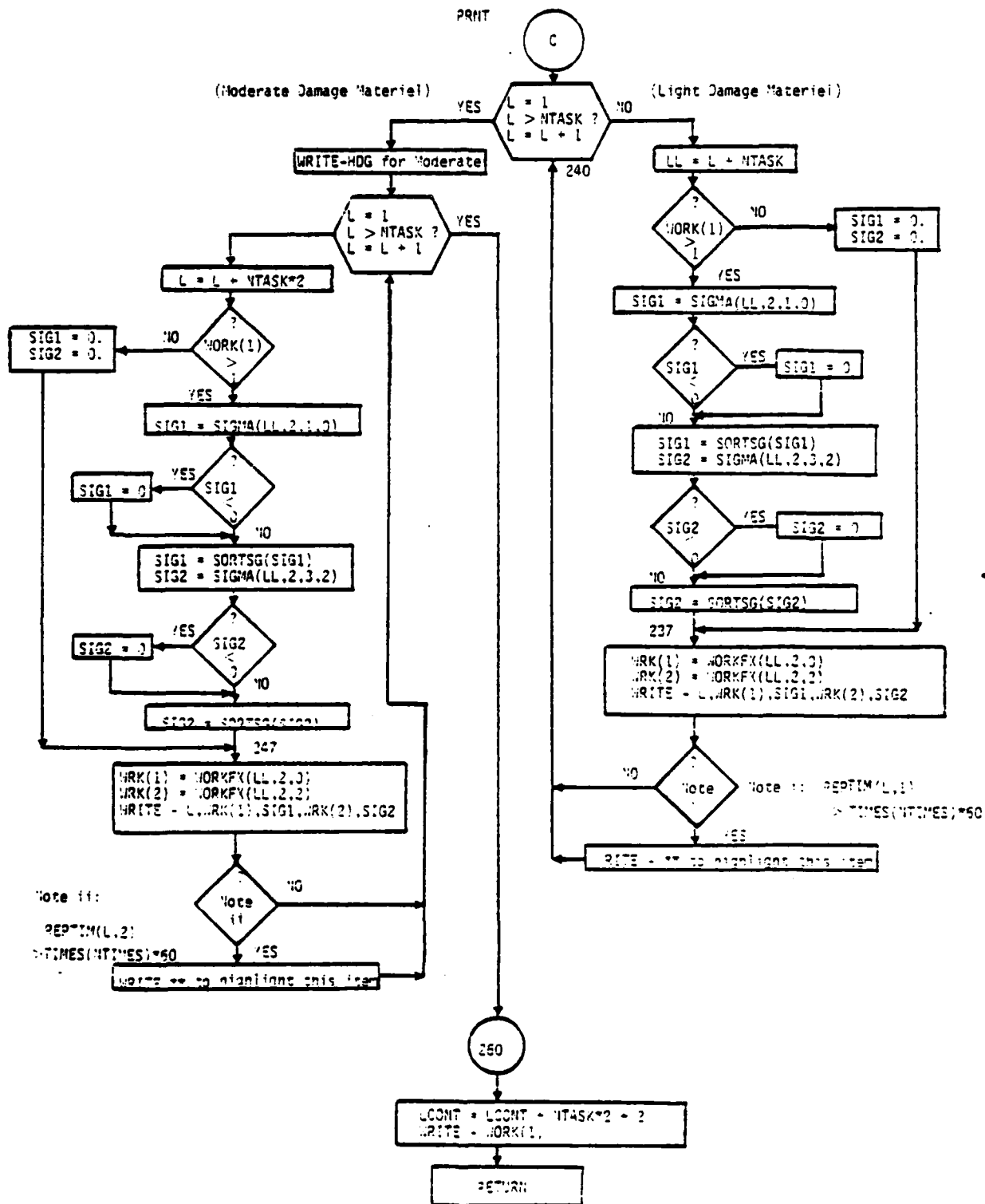


Figure 3-16. Subroutine PRHT (Continued)

3.17 SUBROUTINE OUTA

3.17.1 General

Subroutine OUTA is called by the MAIN routine if the option flag ASSIGN is greater than zero. Subroutine OUTA controls the calculation and printing of assignment data by appropriate calls to subroutine PRNTA. The routine reads the first element, WORK(1), of each record in DEFINE FILE 22. The read is within a set of nested loops for mission number, personnel/materiel, and team number. The first element of each record is the count of the number of iterations for which data was accumulated in that record. Subroutine PRNTA is called when this element is greater than zero. The calling arguments are: J1, mission number; K1, personnel or materiel; L1, team number; and KOUNT, the record number found to have data.

3.17.2 COMMON BLOCKS

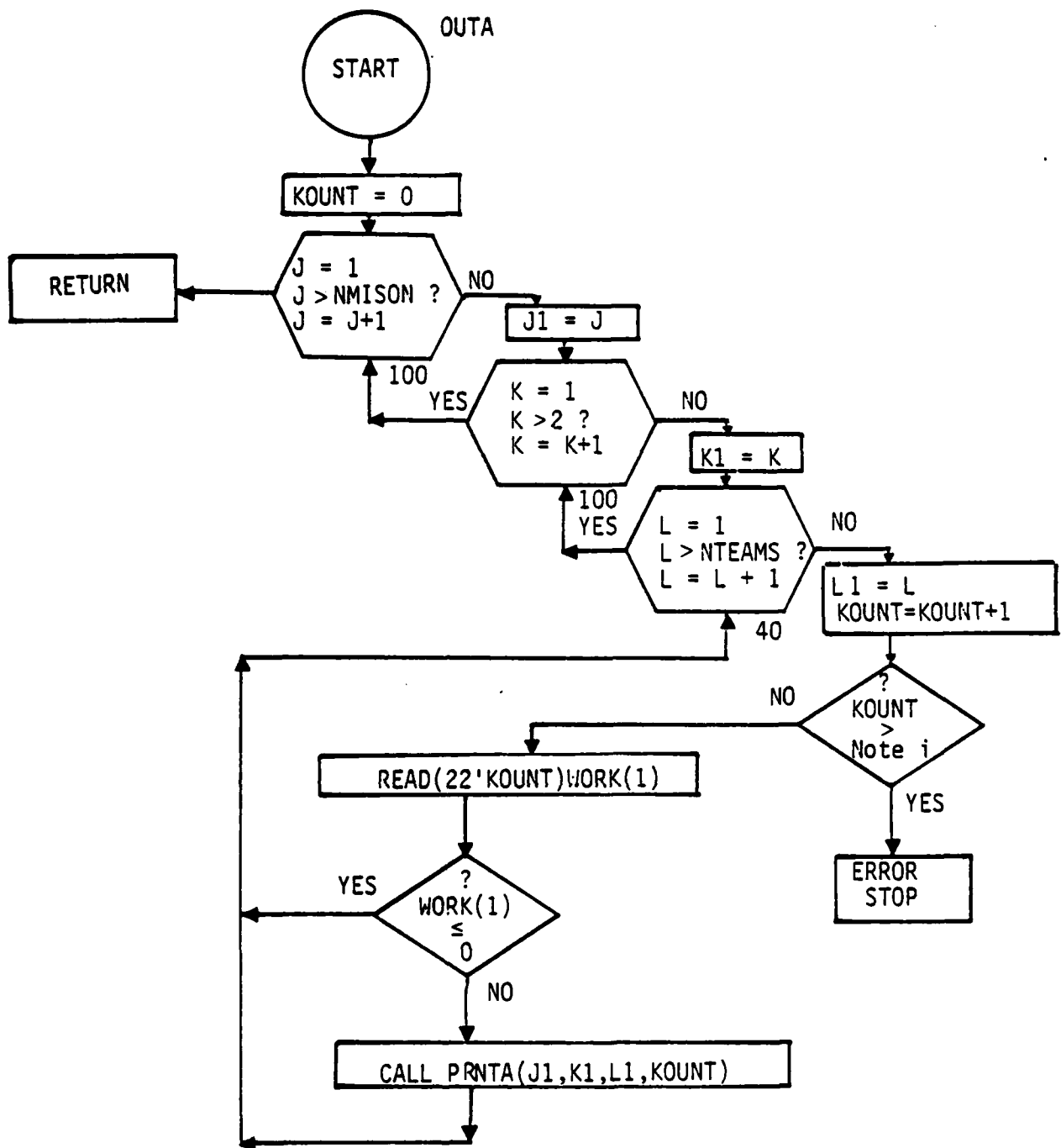
GENERL
PRNTIT
WK2

3.18 SUBROUTINE PRNTA (J, K, L, KOUNT)

3.18.1 General

Subroutine PRNTA is called by subroutine OUTA for the calculation and printing of assignment data. Assignments of each item to fill each requirement (the allocations made by the transportation algorithm to build a particular team) are recorded in DEFINE FILE 22 by subroutine ASN (para 3.11).

The calling arguments furnished by OUTA are: J, mission number; K, personnel/materiel; L, team number; and KOUNT, the record



Note i: $NTEAMS * 2 * NMISON$

Figure 3-17. Subroutine OUTA

number, in DEFINE FILE 22 to be read. The data in DEFINE FILE 22 is accumulated for all iterations that a particular mission-team was the maximum and represents the allocation of each particular item to the fill of a requirement or to surplus. The first element of each record is the number of iterations represented by the data. The record is read into array WORK and element WORK(1) is used to calculate average values for all other elements, the assignments. (NOTE: The element WORK(2) is not used, assignment data for Task 1 begins in element 3, WORK(3).) The average survivors for each task line are accumulated into array PS to get totals.

The calculations and printing are made by task line for a maximum of thirteen columns plus surplus and total. IFIRST and ILAST define the limits for this procedure and only when ILAST is equal to or greater than the numbers of tasks in the surplus and total for that task printed. The variable IGM is used to indicate if a particular row does have some value greater than zero in it. If all values in a row are zero that row is not printed, except when the surplus and total are to be printed.

Three sets of nested loops perform the calculations and the output. The first set is used for both personnel and undamaged materiel. The second and third set perform the same functions but require different indexes for the assignment of materiel items in the light and moderate damage categories.

3.18.2 COMMON BLOCKS

GENERL
PRNTIT
WK2

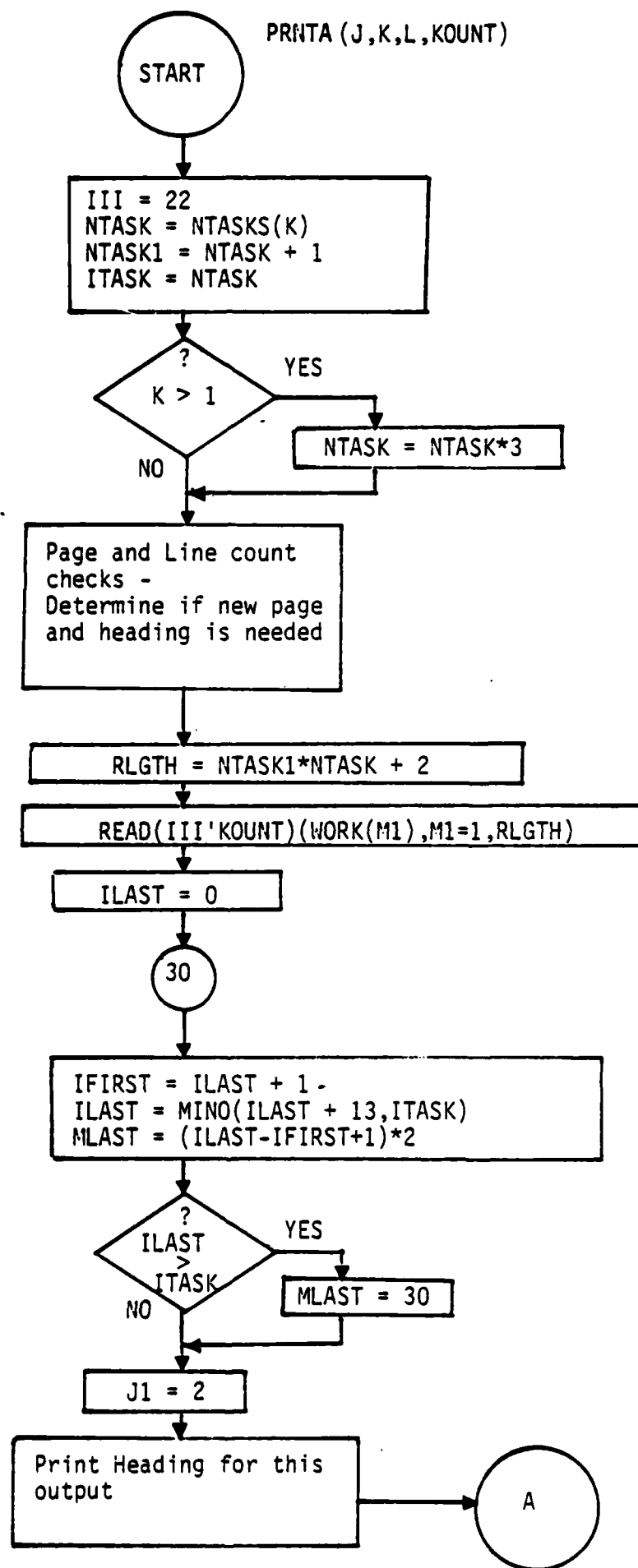


Figure 3-18. Subroutine PRNTA

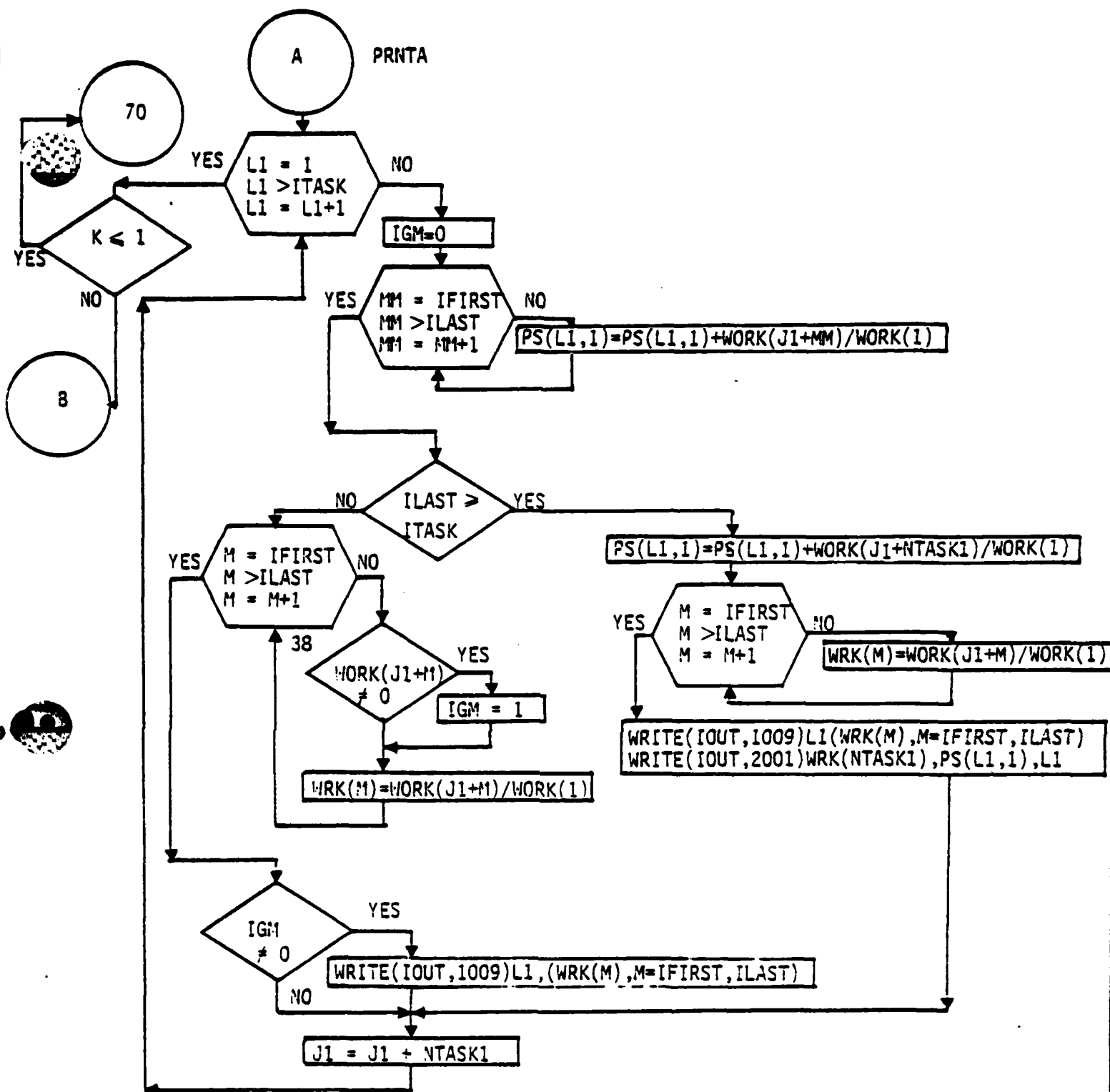


Figure 3-18. Subroutine PRNTA (Continued)

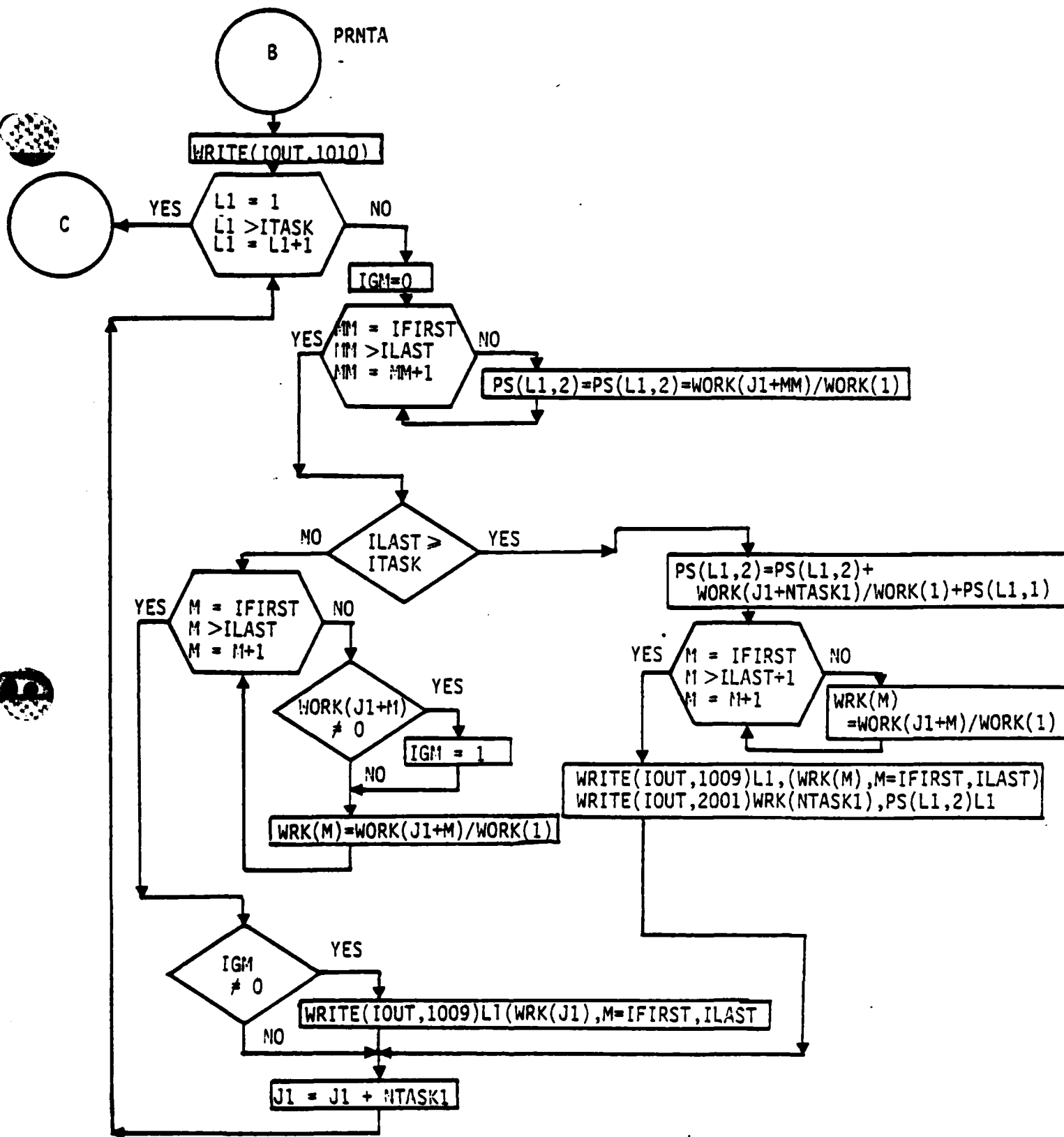
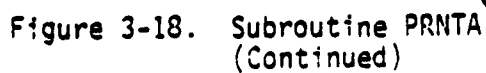


Figure 3-18. Subroutine PRNTA (Continued)



3.19 PROGRAM PARAM

3.19.1 General

Program PARAM is used to read any input file and construct a file of PARAMETER statements which define all the dimensioning variables required by Program AMORE. These dimensioning variables must be made available to the AMORE routine through the FORTRAN PROC element GPARAM.

Program PARAM reads the AMORE input deck to obtain six of the input variables. Those variables are: (1) number of times, NTIMES; (2) number of personnel tasks, NTASKS(1); (3) number of materiel lines, NTASKS(2); (4) total number of authorized individual personnel and materiel items, RANDS; (5) number of teams, NTEAMS; and (6) number of missions, NMISON. These variables are then used to calculate a total of nineteen dimensioning variables. These variables are fully defined in paragraph 2.2, Chapter 2.

PARAM reads from a file designated unit 5. The output is written to a file designated unit 10. This output file must then be transferred to a user designated file/element if retention is desired. The Procedure Definition Processor (PDP) must then be exercised on this output to create the FORTRAN PROC.

3.19.2 Operation

A typical runstream is shown below:

```
@RUN
@ASG,A PROGFILE.
@ASG,A DATAFILE.
@ASG,T 10.
@XQT PROGFILE.PARAM/Abs
@ADD DATAFILE.GPARAM
@ED 10., PROGFILE.GPARAM
@PDP,FL PROFGILE.GPARAM, .GPARAM
@FIN
```

The FORTRAN PROC, GPARAM is now available for the compile
of the AMORE model.

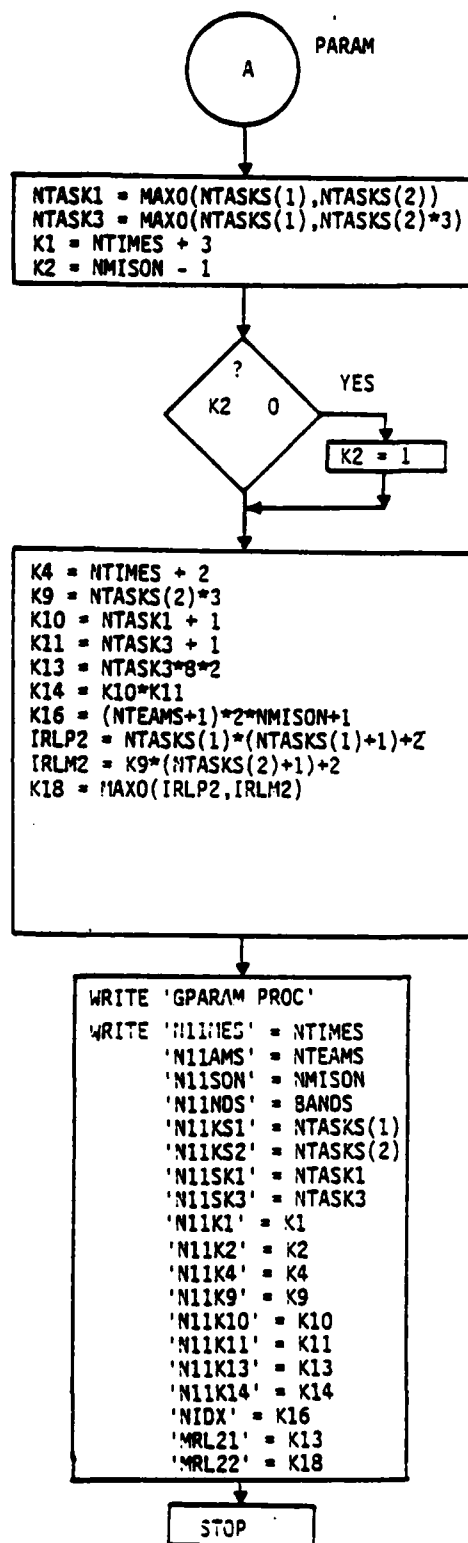


Figure 3-19. Program PARAM (Continued)

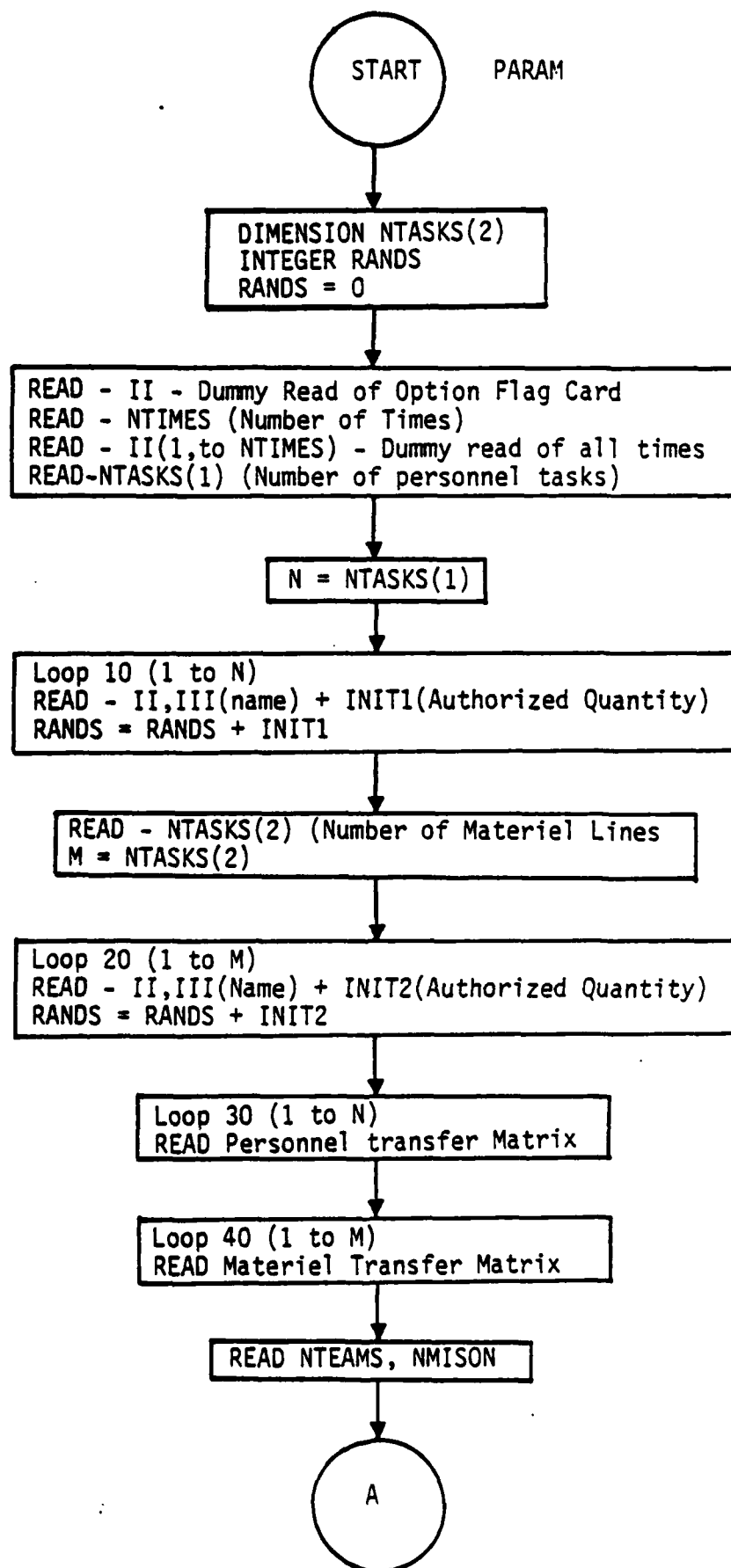


Figure 3-19. Program PARAM

SECTION 4

OPERATING ENVIRONMENT

4.1 HARDWARE

The AMORE model is operational on the UNIVAC 1100/82. The model has no unusual requirements. Input can be either in batch card form or from stored files of card images. Output requires a standard line printer.

CPU Requirements

As noted in paragraph 2.2, the required dimensions for various arrays within the model are determined by six of the input variables. These variables may be used to define a total of 19 parameters which must be in a FORTRAN PROC, "GPARAM". This PROC is INCLUDE'd in each component of the model. The program "PARAM" (para 3.19) provides the capability to construct this PROC file to fit the requirements of any input deck.

The model requires an IBANK of 12632 decimal words. The following equation can be used to determine the DBANK requirement for a given set of dimension variables as defined in paragraph 2.2.

$$N11NDS + 2(N11KS1) + 2(N11KS1)^2 + 6(N11KS2) + 4(N11KS2)^2 \\ + 20(N11SK1) + 21(N11SK3) + (N11SK1 \times N11MES) + 4(N11SK1 \times N11SK3)$$

If Missions = 1

$$+ 12(N11MES) + N11AMS(N11KS1 + N11KS2) + 10614$$

Or if Mission ≥ 2

$$+ (N11AMS \times N11SON)(N11KS1 + N11KS2) + 10(N11MES \times N11SON) \\ + 30(N11SON) + 10578$$

The DBANK required if the model is dimensioned for the example unit in the Users Manual is 27050 decimal words. The input for that unit had 24 times, 35 personnel tasks, 19 materiel lines, 375 individual personnel and materiel items, 18 teams, and 1 mission.

4.2 SUPPORT SOFTWARE

The program is written in ANSI FORTRAN and the UNIVAC ASCII FORTRAN compiler @FTN must be used for compilation. The model uses the random number generator BARN which is included in the CAA LIB\$*FTN. Transfer of the model to other facilities should ensure the availability of a compatible random number generator.

APPENDIX A

MUNKRES' ALGORITHM

A.1 GENERAL

The simplest case of the general allocation problem is the assignment problem. It is merely a situation which involves the assignment of n available objects to n points of need, where some cost accompanies each assignment. For example, a unit may require certain skills to perform three tasks. If there are only three personnel with these skills remaining in this unit, the unit commander would probably make an estimate of how much time it would take for each of these skilled personnel to move to each task location and be prepared to perform the tasks. His estimate of those times can be expressed in the cells of a matrix as shown in Figure A-1.

		TASKS (DEMANDS)		
		I	II	III
SKILLED PERSONNEL (RESOURCES)	A	20 min	40 min	30 min
	B	10 min	30 min	40 min
	C	20 min	10 min	40 min

FIGURE A-1.

The commander would likely wish to have the skills assigned as quickly as possible, in order to reach full capability in minimum time. Therefore, the total assignment time is to be kept to a minimum.

The optimal solution is as follows:

Skill A to Task III Time = 30 min.

Skill B to Task I Time = 10 min.

Skill C to Task II Time = 10 min.

Total time to assign skills is 50 min.

The transportation problem, is, in difficulty, only one degree removed from the assignment problem. Instead of n objects to be sent to n locations, the transportation problem involves n source points and m destination points. Munkres' algorithm is a highly efficient solution technique for the transportation problem which assigns specific resources to meet specific demands in a manner such that either a minimization or a maximization of cost may be obtained. In the AMORE model, a minimum optimal solution in terms of time is sought using Munkres' algorithm. Specifically, the demands are the mission requirements needed to incrementally build capability following some form of degradation. The resources are the surviving personnel and materiel that can be assigned to satisfy the demands. The costs are the times to transfer these resources, the delay time in the decision making process, and in the case of damage materiel items, repair times. If all resources are assigned and all demands are satisfied, then a solution has been found. The cost of a solution is then the sum of the cost of all assignments. An optimal minimum solution is one for which this sum has the lowest possible value. The AMORE process seeks a solution via Munkres' Algorithm, which minimizes the total cost (in terms of time) of the assignments and therefore minimizes the average time per assignment for unit reconstitution.

A.2 ALGORITHM OPERATIONS

Munkres' algorithm first operates on the cost matrix to identify efficient initial assignments. If these initial assignments result in assignment of all resources and satisfaction of all demands, then an optimal solution has been reached. If all assignments have not been made, then Munkres' algorithm repeatedly modifies the cost matrix and makes assignments until optimality is reached. These subsequent assignments will be of two types: (1) a direct assignment of unexhausted row resources to unsatisfied column demands, or (2) reallocations of previous assignments to allow new resource allocations providing minimum cost per assignment made.

A.2.1 Algorithm Steps

Munkres' algorithm, as used in the AMORE process, consists of five basic steps as outlined below. A solution may require one or more of steps two through five to be replicated. Figure A-2 provides a simplified flowchart of the algorithm processes.

- STEP 1 Find the minimum cost in each column and subtract that cost from each cost in the column. Repeat the same procedure for each row. This results in at least one zero cell in each row and column. Go to Step 2.
- STEP 2 Make the maximum allocation of resources possible through the zero cells in the modified matrix. Adjust row resources and column demands accordingly. Cover (flag) those columns where demands have been fully met (zeroed). If all columns are covered, the solution is optimal; otherwise go to Step 3.
- STEP 3 Choose an uncovered zero cell and flag it plus. If it lies in a row with unexhausted resources, label it Z_0 and

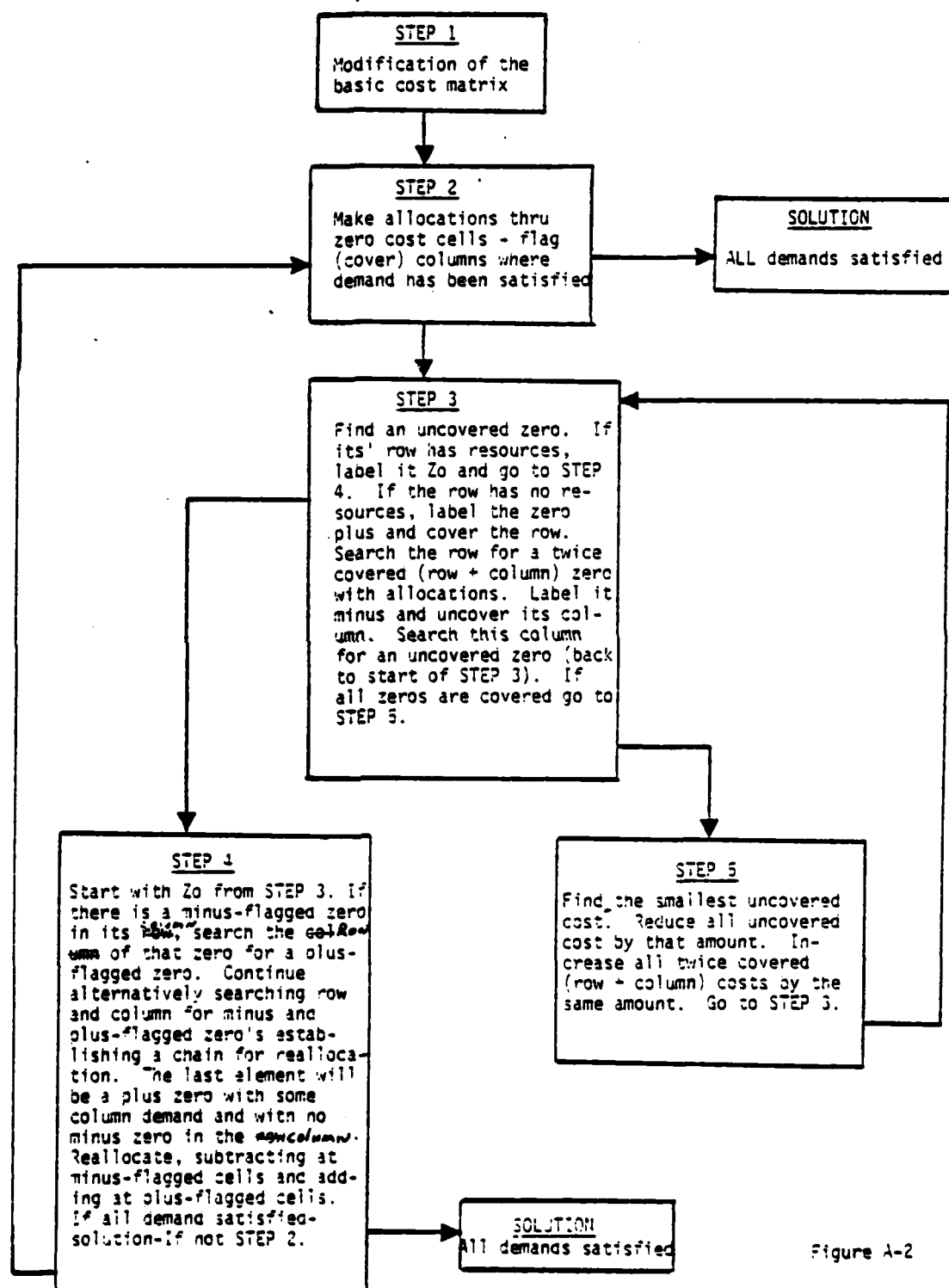


Figure A-2

Figure A-2. Munkres Algorithm

go to Step 4. Otherwise cover the row. If other zeros in the row lie in a covered column and are in a cell having an allocation, flag these zero(s) minus and uncover the associated columns. If all zeros are covered go to Step 5, otherwise repeat step 3.

STEP 4 Beginning with Z_0 search for a minus flagged zero in the column. If none is found, allocate the Z_0 cell the minimum value of either the unallocated resources of the Z_0 row or the unsatisfied demands of the Z_0 column, decrease the Z_0 resources and demands accordingly, remove all flags and covers and go to Step 2. If a minus-flagged zero is found in the column of Z_0 then the row of that zero is searched for a plus-flagged zero. This search procedure is continued, column then row, forming a chain of plus and minus-flagged zero's. The chain ends when a plus-flagged zero is found with no minus-flagged zero in its column. Next, determine which value among the following is minimum: the unallocated resources of the Z_0 row, or the unsatisfied demand of the last plus zero in the chain, or the minimum allocation through the minus zero cell(s). Allocate the resultant minimum value to all plus zero cells in the chain. Decrease all allocations to minus zero cells in the chain. Remove all flags and covers and go to Step 2.

STEP 5 Identify the least uncovered cost, subtract it from all uncovered costs, and add this value to all twice covered costs. Retain all flags and covers and go to Step 3.

A.2.2

An Example Application

A better understanding of the previously discussed five steps of Munkres' algorithm can be gained by means of an example application. Considering the initial cost matrix shown in Figure A-3, the following sequence of algorithm steps derives an optimal solution for

the allocation problem. Note that in the cost matrix, R represents resources available and D represents unsatisfied demands. Each cell contains the cost for satisfying one unit of demand with one unit of resource.

R \ D					
	3	3	3	3	4
1	29	16	19	31	20
5	5	21	15	28	33
7	30	0	22	13	33
3	5	12	32	14	29

FIGURE A-3. THE INITIAL COST MATRIX

Step One: Modification of the Cost Matrix for Initial Assignment. To find an efficient initial assignment, Munkres' algorithm makes use of the following mathematical principle: if a constant is added or subtracted from any row or column of a cost matrix, then an assignment set which minimizes total costs in the new matrix also minimizes total costs in the original cost matrix. The algorithm: Find the minimum cost in the first column. Subtract that cost from each cost in the column. Repeat for all columns. Follow the same procedure for each row. Proceed to Step Two.

Step One produces a new matrix (Figure A-4) with at least one zero in every row and every column. Maximal assignments of resources in Step Two to these zero cells will be efficient. If such an assignment process exhausts all resources, the solution will be optimal.

R \ D	3	3	3	3	4
1	24	16	4	18	0
5	0	21	0	15	13
7	25	0	7	0	13
3	0	12	17	1	9

FIGURE A-4. COST MATRIX FOLLOWING STEP ONE

Step Two: Make the Maximal Initial Assignment to the Modified Cost Matrix: For each zero cell, "remaining" row resources and column demands are compared. The lesser of the two is assigned to the cell and the row resources and column demands reduced accordingly. After all such assignments are made, all columns with satisfied demands are flagged or "covered". If all columns are covered (all demands satisfied), the solution is optimal. The algorithm: Make the maximum allocation through the zero cells in the modified cost matrix. Adjust row resources and column demands accordingly. Cover all columns whose demands have been satisfied. If all columns are covered, an optimal solution has been found. Otherwise, proceed to Step Three. Upon completion of Step Two where a solution is not achieved, the "covering" process paves the way for identifying further opportunities for allocation or reallocation (Figure A-5). The values within the "triangled" upper right-hand corners of zero cells represent allocations of resources assigned to these cells.

R \ D	COVER		COVER		
	0	0	1	0	3
0	24	16	4	18	0 ¹
0	0 ³	21	0 ²	15	13
1	25	0 ³	7	0 ³	13
3	0	12	17	1	9

FIGURE A-5. COST MATRIX FOLLOWING STEP TWO

Step Three: Create a Basis for Further Allocation or Reallocation.

There may be some zeros which are not covered after Step Two. These uncovered zeros will be associated with columns whose demands are not met and rows whose resources are exhausted. Otherwise Step Two would not have been complete (i.e., either additional initial assignments could have been made, or the column should have been covered). Step Three will search for such uncovered zeros to be flagged plus, their rows will then be covered and searched for zeros which are now covered twice (row and column). If there has been an allocation to the cell of a twice covered zero it is flagged minus and its associated column is uncovered. This may uncover some previously covered zeros. This flagging process begins to map out a potential path for allocation or reallocation. The Step Three process ends with one of two conditions. An uncovered zero is identified (to be flagged plus) in a row with resources remaining. Such a zero is identified as Z_0 for allocation and processing now branches to Step Four. Alternatively, all zeros are found to be covered and processing branches to Step Five to modify the cost matrix to create other zeros. The algorithm: Choose an uncovered zero. Flag it as plus. If it lies in a row with unexhausted resources, label it Z_0 and proceed to Step Four. Otherwise, cover the row. If other zeros in this row lie in a covered column and are associated with

a cell that has an allocation, flag the zero minus and uncover its associated column. Repeat the process until an uncovered zero, flagged plus, is found to have resources remaining, then proceed to Step Four. Otherwise, when all zeros are covered go to Step Five.

In applying Step Three to the example Cost Matrix, three iterations of the step are required. Figure A-6 shows the first two iterations as labeling first the (1,5) position and then the (2,3) position with plus zeros. However neither qualify for being labeled Z_0 as both rows have exhausted resources. This results in both rows being covered. However, the latter plus zero row has another zero in it which has an allocation (the (2,1) position) and is therefore labeled minus zero. The column associated with that minus zero is also uncovered. By virtue of uncovering the first column, the third iteration then labels the (4,1) position plus zero and also Z_0 . This latter condition now satisfies the requirements for proceeding to Step Four.

AFTER ITERATIONS 1 AND 2

		COVER		COVER	
R \ D	0	0	1	0	3
0	24	16	4	18	⁺ 0 ¹
0	⁻ 0 ³	21	⁺ 0 ²	15	13
1	25	0 ³	7	0 ³	13
3	0	12	17	1	9

AFTER ITERATION 3

		COVER		COVER	
R \ D	0	0	1	0	3
0	24	16	4	18	⁺ 0 ¹
0	⁻ 0 ³	21	⁺ 0 ²	15	13
1	25	0 ³	7	0 ³	13
3	⁺ 0 ²	12	17	1	9

FIGURE A-6. COST MATRIX FOLLOWING STEP THREE

Step Four: Execute a New Additional Allocation and Any Necessary

Reallocation of Resources to Demands. Step Four begins with a plus-flagged zero designated Z_0 in a row with unexhausted resources. One of two conditions will now apply.

If Z_0 lies in a column with no minus-flagged zeros, then a comparison is made between the remaining row resources and the unsatisfied column demands. The lesser of the two is assigned. After such allocation all plus and minus flags and all row and column coverings are removed. Processing returns to that part of Step Two where all demand satisfied columns are covered and the solution is then tested for completeness.

Alternatively, Z_0 lies in a column with a minus-flagged zero which is designated Z_1 . A search of the row containing Z_1 is made for a plus-flagged zero which is designated Z_2 . The search process is thus continued until a plus-flagged zero is identified which has no minus-flagged zero in its column. Next, a value for allocation is determined. It is the minimum value of three sets considered. Set One is the set of all allocations to cells containing minus-flagged zeros in the above-identified chain. Set Two is the remaining resources in the row containing Z_0 . Set Three is the unsatisfied demand in the column containing the last plus-flagged zero in the chain. The minimum value of the three sets is chosen for reallocation. Allocations previously made through the minus-flagged zeros in the chain are reduced by this amount and each of the plus-flagged zeros in the chain is increased by this amount. The Z_0 row resources and the column demand of the last plus zero in the chain are also reduced by this amount. All plus and minus flags and row and column coverings are removed and processing returns to that part of Step Two which covers all demand satisfied columns and tests the solution for completeness or further processing (Step Three). The algorithm: Beginning with Z_0 search for minus-flagged zeros in the column of a plus zero under consideration,

search for a plus-flagged zero in the row of the identified minus-flagged zero. Continue until arriving at a plus-flagged zero with no minus-flagged zero in its column. (This may occur when Z_0 is the only plus zero.) Identify as an allocation the minimum of the allocations to all minus-flagged zero cells, the row resource at Z_0 , or the column demand of the last plus-flagged zero in the chain. (When Z_0 is the only plus zero the minimum of its corresponding row resources or column demands is used.) Add this minimum value as the allocation to all plus-flagged zeros in the chain; subtract it from all minus-flagged zeros in the chain. Adjust remaining Z_0 row resources and the unsatisfied last plus zero column demands accordingly. If all demands are satisfied, solution is complete. Remove all plus and minus flags and coverings. If demands are not satisfied, return to that part of Step Two to cover all satisfied demand columns.

The chain established by completing Step Three consists of the Z_0 in position (4,1), the minus zero in position (2,1) and the plus zero in position (2, 3). Applying Step Four to the left Cost Matrix of Figure A-7, the minimum value among the Z_0 row resources, the minus zero allocation(s) and the plus zero column demands in the chain is the value of plus zero column demand (value = 1). By allocating this amount to the Z_0 cell and by reducing the minus zero(s) in the chain, the Z_0 row resource and the last plus zero column demand by the same amount, the results are as depicted in the right Cost Matrix of Figure A-7. Note that the flags and covers are also removed upon completion of Step Four prior to returning to Step Two.

BEFORE STEP FOUR

		COVER			COVER		
		D	0	0	1	0	3
Cover	R	0	0	1	0	3	
	0	24	16	4	18	$+0/1$	
	1	$-0/3$	21	$+0/2$	15	13	
	1	25	$0/3$	7	$0/3$	13	
	3	$+0/2$	12	17	1	9	

AFTER STEP FOUR

		D				
		0	0	0	0	3
R	0	24	16	4	18	0/1
	0	0/2	21	0/3	15	13
	1	25	0/3	7	0/3	13
	2	0/1	12	17	1	9

FIGURE A-7. COST MATRIX FOLLOWING THE STEP FOUR PROCEDURE

Returning to Step Two the appropriate portion of the algorithm is repeated: Cover all columns whose demands have been satisfied. The Cost Matrix becomes that shown in Figure A-8 before proceeding to Step Three.

		COVER COVER COVER COVER				
D						
R		0	0	0	0	3
0		24	16	4	18	0 ¹
0		0 ²	21	0 ³	15	13
1		25	0 ³	7	0 ³	13
2		0 ¹	12	17	1	9

FIGURE A-8. COST MATRIX FOLLOWING STEP TWO A SECOND TIME

Proceeding to Step Three again and the algorithm: Choose an uncovered zero. Flag it as plus. If it lies in a row with unexhausted resources, label it Z_0 and proceed to Step Four. Otherwise, cover the row. If other zeros in this row lie in a covered column and are associated with a cell that has an allocation, flag the zero minus and uncover its associated column. Repeat the process until an uncovered zero, flagged plus, is found to have resources remaining. Then proceed to Step Four. Otherwise, when all zeros are covered go to Step Five.

Since all zeros are covered in the Cost Matrix (Figure A-9), Step Five is next.

		COVER COVER COVER COVER					
		D					
		R	0	0	0	0	3
Cover	0	24	16	4	18	+0	1
	0	0 ²	21	0 ³	15	13	
	1	25	0 ³	7	0 ³	13	
	2	0 ¹	12	17	1	9	

FIGURE A-9. COST MATRIX BEFORE ENTERING STEP FIVE

Step Five: Modify the Cost Matrix to Produce New Uncovered Zeros:
All plus and minus flags and row and column coverings are retained. Considering covered rows and columns, there are three sets of cells: uncovered cells, single-covered cells, and twice-covered cells. The minimum cost in the uncovered cells is identified. It is subtracted

from costs in all uncovered cells creating at least one new zero cost in an uncovered cell. It is added to costs in all twice-covered cells. Costs in single-covered cells remain unchanged. Optimal allocations on these adjusted costs will also be optimal for the original cost matrix. Processing returns to Step Three. The algorithm: Identify the least uncovered cost. Subtract this value from all uncovered costs. Add this value to all twice-covered costs. Retain all plus and minus flags and cover and return to Step Three.

The least uncovered cost in Figure A-9 is that found in position (4,5) valued nine. Subtracting this from uncovered positions (2,5), (3,5) and (4,5) and adding to twice covered positions (1,1), (1,2), (1,3) and (1,4) results in the cost matrix shown in Figure A-10.

		COVER COVER COVER COVER					
		D	0	0	0	0	3
COVER	R						
	0	33	25	13	27	0 ⁺	1
	0	0 ²	21	0 ³	15		4
	1	25	0 ³	7	0 ³		4
	2	0 ¹	12	17	1		0

FIGURE A-10. COST MATRIX FOLLOWING STEP FIVE

Again returning to Step Three and that part of the algorithm which pertains: Choose an uncovered zero. Flag it as plus. If it lies in a row with unexhausted resources, label it Z_0 and proceed to Step Four (Figure A-11).

COVER COVER COVER COVER							
COVER		D	0	0	0	0	3
		R	0	0	0	0	3
COVER	0	0	33	25	13	27	$+0^1$
	0	0	0^2	21	0^3	15	4
	1	1	25	0^3	7	0^3	4
	2	2	0^1	12	17	1	$+0^2$

FIGURE A-11. COST MATRIX FOLLOWING STEP THREE

Following that part of the Step Four algorithm which pertains: Beginning with Z_0 , search for minus-flagged zeros in the column of a plus zero under consideration, search for a plus-flagged zero in the row of the identified minus-flagged zero. Continue until arriving at a plus-flagged zero with no minus-flagged zero in its column. (This may occur when Z_0 is the only plus zero.) Identify as an allocation (in this case) the minimum of either the Z_0 row resources or the Z_0 column demands. Add this minimum value as the allocation to all plus-flagged zeros (Z_0) in the chain; and adjust remaining Z_0 row resources and the Z_0 column demands accordingly. If all demands are satisfied, solution is complete. Remove all plus and minus flags and coverings. If demands are not satisfied, return to that part of Step Two to cover all satisfied demand columns. This results in Figure A-12.

	R \ D	0	0	0	0	1
	0	33	25	13	27	0^1
	0	0^2	21	0^3	15	4
	1	25	0^3	7	0^3	4
	0	0^1	12	17	1	0^2

FIGURE A-12. COST MATRIX FOLLOWING STEP FOUR

Again repeating the pertinent parts of Step Two's algorithm: Cover all columns whose demands have been satisfied. The Cost Matrix before proceeding to Step Three is as shown in Figure A-13.

		COVER COVER COVER COVER				
D \ R		0	0	0	0	1
	0	33	25	13	27	0 ¹
	0	0 ²	21	0 ³	15	4
	1	25	0 ³	7	0 ³	4
	0	0 ¹	12	17	1	0 ²

FIGURE A-13. COST MATRIX FOLLOWING STEP TWO

Once more returning to Step Three and executing the algorithm: Choose an uncovered zero. Flag it as plus. If it lies in a row with unexhausted resources, label it Z_0 and proceed to Step Four. Otherwise, cover the row. If other zeros in this row lie in a covered column and are associated with a cell that has an allocation, flag the zero minus and uncover its associated column. Repeat the process until an uncovered zero, flagged plus, is found to have resources remaining, then proceed to Step Four. Otherwise, when all zeros are covered go to Step Five.

After three iterations of Step Three the Cost Matrix results in Figure A-14 with all zeros covered. This then requires proceeding to Step Five.

		COVER		COVER	
COVER	D	0	0	0	1
	R				
COVER	0	33	25	13	27 ⁺⁰ / ₁
COVER	0	⁺⁰ / ₂	21	⁻⁰ / ₃	15 4
	1	25	⁰ / ₃	7	⁰ / ₃ 4
COVER	0	⁻⁰ / ₁	12	17	1 ⁺⁰ / ₂

FIGURE A-14. COST MATRIX FOLLOWING STEP THREE

Repeating the Step Five Algorithm: Identify the least uncovered cost. Subtract this value from all uncovered costs. Add this value to all twice-covered costs. Retain all plus and minus flags and cover and return to Step Three.

The least uncovered cost in Figure A-14 is that (value = 4) found in position (3,5). Subtracting this value from positions (3,1), (3,3) and (3,5) and adding it to twice covered positions (1,2), (1,4), (2,2), (2,4), (4,2), and (4,4) results in the Cost Matrix shown in Figure A-15.

		COVER		COVER	
COVER	D	0	0	0	1
	R				
COVER	0	33	29	13	31 ⁺⁰ / ₁
COVER	0	⁺⁰ / ₂	25	⁻⁰ / ₃	19 4
	1	21	⁰ / ₃	3	⁰ / ₃ 0
COVER	0	⁻⁰ / ₁	16	17	5 ⁺⁰ / ₂

FIGURE A-15. COST MATRIX FOLLOWING STEP FIVE

Returning to Step Three again to execute only the pertinent part of the algorithm: Choose an uncovered zero. Flag it as plus. If it lies in a row with unexhausted resources, label it Z_0 and proceed to Step Four. This results in the Cost Matrix depicted in Figure A-16.

		COVER		COVER	
COVER	R \ D	0	0	0	1
	0	33	29	13	31
COVER	0	+0/2	25	-0/3	19
	1	21	0/3	3	0/3
COVER	0	-0/1	16	17	5

FIGURE A-16. COST MATRIX FOLLOWING STEP THREE

As with a previous execution of Step Four only the pertinent parts of the algorithm follow: Beginning with Z_0 search for minus-flagged zeros in the column of a plus zero under consideration, search for a plus-flagged zero in the row of the identified minus-flagged zero. Continue until arriving at a plus-flagged zero with no minus-flagged zero in its column. (This may occur when Z_0 is the only plus zero.) Identify as an allocation the minimum of either the Z_0 row resources or the Z_0 column demands. Add this minimum value as an allocation to all plus-flagged zeros in the chain; and adjust remaining Z_0 row resources and the Z_0 column demands accordingly. If all demands are satisfied, solution is complete. Remove all plus and minus flags and coverings. If demands are not satisfied, return to that part of Step Two to cover all satisfied demand columns. This results in the complete solution (Payoff Matrix) shown in Figure A-17.

R \ D					
	0	0	0	0	0
0	33	29	13	31	0 ¹
0	0 ²	25	0 ³	19	4
0	21	0 ³	3	0 ³	0 ¹
0	0 ¹	16	17	5	0 ²

FIGURE A-17. SOLUTION PAYOFF MATRIX FOLLOWING THE LAST ITERATION OF STEP FOUR

Now that the solution is complete, the cost must be determined. By projecting the solution allocations onto the original Cost Matrix the matrix shown in Figure A-18 results.

29	16	19	31	20 ¹
5 ²	21	15 ³	28	33
30	0 ³	22	13 ³	33 ¹
5 ¹	12	32	14	29 ²

FIGURE A-18. THE FINAL COST AND ALLOCATION MATRIX

By multiplying each cell cost by the corresponding allocation to that cell and summing, the optimum (minimum) total cost is determined as follows:

$$(1 \times 20) + (2 \times 5) + (3 \times 15) + (3 \times 0) + (3 \times 13) + (1 \times 33) + (1 \times 5) + (2 \times 29) = 210$$

A.3 ALTERNATE OPTIMAL SOLUTIONS

The basic Munkres Algorithm provides an optimal solution to the allocation problem and also provides a convenient means for determining if other combinations of allocations exist which would also be optimal. The modified cost matrix (called the payoff matrix) which results from the solution of the basic problem provides this means.

If a resource allocation which has been made is applied to a different demand, an equal amount of some previous allocation against that demand must be reallocated, so that all demands are exactly met and all resources are allocated. This process is repeated forming a chain through the matrix. In order to keep the resource and demand in balance, this reallocation "chain" must eventually reallocate an equal amount of supply to the demand from which the original reallocation was made. The reallocations made through this "chain" must be equal; therefore, the largest possible reallocation which can be made will be equal to the smallest of the original allocations.

The payoff matrix which resulted from the original solution will always have zero values in cells where allocations have been made. There may also be other zero values in the matrix which represent possible opportunities for reallocations. If a chain (complete loop) of these zero values can be found, a reallocation made through that chain will also be an optimal solution.

The process is as follows: Pick an allocation. Since it is desired to move some of this allocation (subtract), this cell is flagged with a minus. Now examine the payoff matrix row of that allocation for a payoff value of zero ($P = 0$). Because we intend to add allocations to this cell it is flagged as a plus. In the column of this $P = 0$, find another $P = 0$, this cell must have a previous allocation since we

must subtract from it and negative allocations cannot exist, this cell is flagged minus. Examine the row of this cell for another $P = 0$ and flag it plus. Continue this process until the chain returns to the original allocation cell. Next examine all minus flagged cells to find the smallest value of the allocations made to those cells. This is the largest possible reallocation which can be made. (Any smaller reallocation could also be made to provide an intermediate solution.) This value is then subtracted from all previous allocations in the minus flagged cells and added to the allocations in all plus flagged cells. This completes the reallocation and results in a different but still optimal solution.

An examination of the final payoff matrix from the previous example problem (Figure A-18) shows that no alternate solution is possible. Therefore, an example of an alternate solution is provided by the cost matrix shown in Figure A-19.

R \ D				
	15	10	4	1
10	6	5	4	3
10	2	1	5	6
10	3	2	1	2

FIGURE A-19. ORIGINAL COST MATRIX

Applying the five steps of Munkres' Algorithm to this cost matrix the solution Payoff Matrix shown in Figure A-20 is derived.

0/9	0	0	0/1
0/1	0/9	5	7
0/5	0/1	0/4	2

TOTAL COST = 89

FIGURE A-20. THE PAYOFF MATRIX

Although a number of reallocation combinations can be derived from the example payoff matrix, the following development of a reallocation chain presents a good representative sample for discussion purposes. Beginning by selecting position (1,1) since it has an allocation, it is flagged minus (Figure A-21).

-0/9	0	0	0/1
0/1	0/9	5	7
0/5	0/1	0/4	2

FIGURE A-21

Examining the row containing this minus zero cell we find two zero cells without allocations. For example purposes, the one located at position (1,3) is selected and flagged as a plus zero (Figure A-22).

-0/9	0	+0	0/1
0/1	0/9	5	7
0/5	0/1	0/4	2

FIGURE A-22

Now, searching the column of the plus zero, another zero with an allocation is found (position (3,3)). This is labeled with a minus (Figure A-23).

-0/9	0	+0	0/1
0/1	0/9	5	7
0/5	0/1	-0/4	2

FIGURE A-23

Next a search is made of this minus zero row for another zero cell. Assuming the one at position (3,2) is selected, it is flagged plus (Figure A-24).

-0/9	0	+0	0/1
0/1	0/9	5	7
0/5	+0/1	0/4	2

FIGURE A-24

Again searching the column of the last plus flagged zero for a zero cell with an allocation, one is located at position (2,2). This cell is flagged minus (Figure A-25).

-0/9	0	+0	0/1
0/1	-0/9	5	7
0/5	+0/1	-0/4	2

FIGURE A-25

Searching the row of this minus zero cell reveals another zero cell at position (2,1). Labeling this zero cell plus and searching its column, the chain is found to be complete as the initial minus flagged zero cell lies in this column (Figure A-26).

-0/9	0	+0	0/1
+0/1	-0/9	5	7
0/5	+0/1	-0/4	2

FIGURE A-26. THE REALLOCATION CHAIN COMPLETED

By examining the minus flagged cells in reallocation chain just developed, the minimum allocation among them is four in position (3,3). By subtracting this amount from the minus flagged cells (positions (1,1), (2,2) and (3,3)) in the chain and adding it to the plus flagged cells (positions (1,3), (2,1) and (3,2)), the reallocation is completed as shown in Figure A-27. Note cell (3,3) now has a zero allocation, and other allocations are also changed but the solution cost remains the same.

0/5	0	0/4	0/1
0/5	0/5	5	7
0/5	0/5	0	2

TOTAL COST = 89

FIGURE A-27. ALTERNATE SOLUTION